UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITECNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA

TRABAJO FIN DE GRADO

APLICACIÓN ANDROID PARA LA SUPERVISIÓN Y CONTROL DE UN SISTEMA DE TELEMONITORIZACIÓN EN TIEMPO REAL

Autor: **Juan Rubio Iradi**

Tutor: **Francisco Javier Ordóñez Morales**

Leganés, Septiembre de 2012

A mis padres y hermanos: José Félix, Maite, Coro y José.

Resumen

Internet ha causado junto con los avances en la telefonía móvil una repercusión considerable en nuestra vida y en como nos comunicamos. Lo que antes podían parecer dos realidades separadas, actualmente se complementan, y de hecho nos parecería raro, incluso, pensar en un móvil sin conexión a Internet en cualquiera de sus formas.

Con el uso de Internet la demanda sobre los teléfonos móviles ha ido creciendo a un ritmo alarmante, y los fabricantes en su empeño por no quedarse atrás han ido invirtiendo más esfuerzo y dinero en mejorarlos, hasta lo que hoy en día se conocen como *smartphones*. De esta forma al ofrecerse un dispositivo con mejores prestaciones, surgen a su vez sistemas operativos más capaces y eficientes, y gradualmente aparecen multitud de aplicaciones que desean participar ofreciendo su granito de arena. Pensando en la movilidad que ofrece un dispositivo de este calibre y en la posibilidad de conectarlo a Internet, surgen ideas tan variadas como herramientas para la salud, mensajería que sustituye al tradicional *sms*, aplicaciones de gestión, e incluso herramientas de telemonitorización, que es justo en este último tipo en el que se centra este Trabajo de Fin de Grado.

Quizás el término telemonitorización se perciba de un modo muy abstracto y vago, por lo que mejor pensemos en algo así como una herramienta de monitorización a distancia. La posibilidad de conectar el móvil a un hogar nos ofrece usos muy variopintos. Supongamos un domicilio en el que residen personas de avanzada edad y que por una razón u otra no las podemos atender presencialmente. La idea sería colocar sensores en dicho domicilio que recogiesen datos del momento de su activación, y que dicho evento llegase a nuestro dispositivo móvil con toda la información que se necesitase. Para ello la información debería recibirse en tiempo real, o al menos en *soft real-time*, en cualquier dispositivo móvil mediante un *push* realizado desde un servidor centralizado.

Así pues, en este proyecto se documenta el desarrollo de una aplicación a modo de *front–end* para el sistema operativo Android que monitorice una red de sensores ya existente. Para dicho fin, el *front–end* anteriormente mencionado, deberá conectarse mediante un protocolo *publish/subscribe* (en nuestro caso MQTT) al servidor encargado de enviar los mensajes. Tras conectarse, deberá ser capaz de suscribirse a un canal específico para la recepción de dichos eventos. Además, la aplicación deberá almacenar las activaciones producidas por cada sensor a modo de registro histórico. Dicho histórico podrá ser usado en un futuro para la generación de gráficas de un sensor dado y según intervalos de tiempo concretos.

La aplicación, desarrollada sobre una arquitectura MVC, ha sido pulida a lo largo de la vida de este TFG solventando diversos problemas para conseguir un sistema que cumple con todas las expectativas del cliente. Todo ello implementado de un modo sencillo y moderno que no requiera de tecnicismos ni conocimientos que un usuario medio no disponga, con un consumo mínimo de recursos, con rendimiento en *soft realtime* y sobre todo, sirviendo de base para el desarrollo de tecnología móvil de telemonitorización.

Abstract

Internet has caused together with mobile telephony progress a huge impact on our lives and the way we communicate. What once might have seemed as two separated realities, are complemented nowadays, and indeed would seem strange even to think of a mobile with no Internet connection in any of its forms.

The more the Internet is used the more the growth at the demand on mobile phones, and the manufacturers in their effort to keep up, have been investing more effort and money to improve them, to what today are known as *smartphones*. Thus by offering a device with better features, more capable and efficient operating systems are emerging at the same time, and gradually appear many applications that wish to participate by offering their two cents. Considering the mobility that offers a device of such a kind and the possibility of connecting it to Internet, ideas emerge as varied as health tools, messaging which replaces the traditional *sms*, management applications, and even telemonitoring tools, which is just in the latter type in which this Final Project Work focuses.

Perhaps the term "telemonitoring" is perceived in a very abstract and vague way, so better think of something like a remote monitoring tool. The ability of connecting a cell phone to a home offers us very different uses. Imagine a home where elder people live and for one reason or another it is not possible to attend in person. The idea would be to place sensors in such address to collect data of the activation time, and that this event came to your mobile device with all the information that was needed. For this purpose, the information will be received at real time, or at least at *soft real-time*, on any mobile device carried by a push from a centralized server.

Hence, this project documents the development of an application that acts as a *front–end* for the Android operating system, monitoring an existing sensor network. To this end, the *front–end* above, will connect through a *publish/subscribe* protocol (in our case MQTT) to the server responsible for sending the messages. After connection established, it will be able to subscribe to a specific channel for the reception of such events. In addition, the application must store the activations produced by each sensor as a historical record. Such record may be used in the future to generate graphs for a given sensor according to specific time intervals.

The application, developed on an MVC architecture, has been "polished" over the life of this Project by solving various problems and finally getting a system that meets all customer expectations. All implemented in a simple and modern way, with no technicalities or knowledge requirement, that an average user does not have, with minimal consumption of resources, with *soft real-time* performance, and above all, serving as the foundation for the development of telemonitoring mobile technology.

Índice general

Índice de figuras

Índice de tablas

1 Introducción

"Podría parecer que hemos llegado a los límites alcanzables por la tecnología informática, aunque uno debe ser prudente con estas afirmaciones, pues tienden a sonar bastante tontas en cinco años".

John Von Neumann (1949)

1.1 Motivación

Es interesante echar la vista atrás y ver como se ha evolucionado, tecnológicamente hablando, en tan pocos años. Ya nos parece lejano recordar los orígenes de Internet (allá por el 1969) e incluso de la telefonía móvil (1980), pero lo cierto es que casi todos los avances significativos en el mundo de la tecnología han ocurrido en un lapso de tiempo muy reciente si lo comparamos con los más de 2 millones de años que lleva el género *Homo* sobre la faz de la tierra.

Esta evolución hace alusión más a una curva exponencial que a una línea recta, que está en constante crecimiento. El continuo crecimiento se debe a que cuando un paradigma deja de ser usable surge otro nuevo para remplazarlo. Un ejemplo claro es la transición que hubo entre los tubos de vacío y los transistores, ya que cuando los primeros llegaron a su límite, estos últimos llegaron para sustituirlos a un precio más bajo, mucho más pequeños y aportando mayor fiabilidad. Los transistores tradicionales se sustituirán por transistores tridimensionales (*Tri-Gate* [1]) y así sucesivamente.
En cuanto a que el crecimiento sea exponencial, Ray Kurzweil [2] lo deja bien claro haciendo particular énfasis en el fenómeno de la *Singularidad* que vaticina para el 2045 aproximadamente (véase la Figura 1), momento a partir del cual es imposible imaginar más allá (al igual que el horizonte de sucesos en un agujero negro o acontecimientos previos al B*ig Bang*), pero bueno..., ese es otro tema.

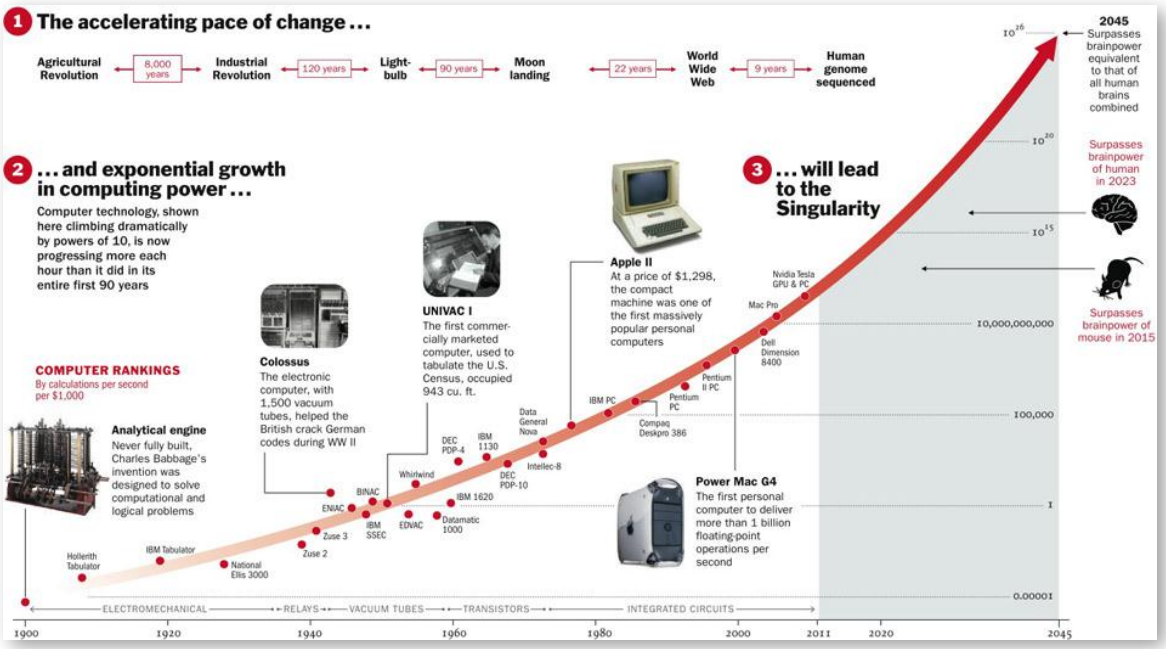


Figura 1. Aceleración tecnológica - Singularidad

Volviendo a lo que se comentaba anteriormente, al igual que con los tubos de vacío y los transistores, la telefonía móvil también ha sido sometida a constante alteración. Ha pasado de ser un aparato electrónico que se empleaba únicamente para realizar llamadas, a algo bastante más complejo denominado "*Smartphone*" o teléfono inteligente. Lo cierto es que actualmente este dispositivo es más parecido a una computadora de bolsillo que a la noción clásica de "teléfono móvil", incluyendo a su vez su propio Sistema Operativo sobre el que poder instalar las aplicaciones o programas que se deseen.

Junto con el auge de Internet, un Smartphone permite ahora cosas tan variopintas como navegar, consultar el correo electrónico, modificar documentos, escuchar música, emplearlo como GPS para el coche, e incluso tener la cartografía del mundo en la palma de la mano.

Esto sin duda alguna da lugar a muchas oportunidades, y no refiriéndose únicamente a que la conectividad y la capacidad de cómputo sean mucho mayores que las de un teléfono convencional, sino que derivado de éste y otros tantos factores, existe una creciente demanda sobre el desarrollo de aplicaciones para este sector [3].

Es esta insaciable demanda la que se desea explotar junto con las ventajas que ofrece la "movilidad" de este tipo de dispositivos,...pero ¿de qué forma se puede aportar a la sociedad mediante este mecanismo de un modo útil?
Supóngase por ejemplo que necesita saber el estado en el que se encuentran ciertas personas que habitan en una determinada casa, por una razón x (es gente anciana, pacientes que necesitan cuidados especiales, ha dejado a sus hijos solos en casa, etc.), pero le es imposible averiguarlo presencialmente. Este proyecto se basa justamente en esa idea, es decir, mediante un simple mecanismo de telemonitorización poder realizar cierto seguimiento sobre las personas a supervisar sin necesidad de acudir a su propio domicilio. Toda la información necesaria se vería reflejada en el mismo *Smartphone* con toda la comodidad, movilidad y ahorro de recursos que esto implica.

En una primera instancia la aplicación Android a desarrollar se conectaría a una red de sensores binarios implantados en dicho alojamiento, pero en una situación futurible, esto podría ser perfectamente escalable a otro tipo de sensores que recopilasen información más compleja, ya que el protocolo de comunicación se mantendría invariable.

1.2 Estructura del documento

Para facilitar la lectura de la memoria, la estructura del resto del documento se detalla a continuación:

1. Introducción:

   Se realiza la introducción aportando cual es la motivación del proyecto, sobre que trata, cual es el problema que se desea resolver y un breve desglose del resto del documento.

2. Gestión y Organización del proyecto:

   En esta sección se detalla la metodología que se va a seguir en la ejecución de este proyecto.

3. Estado de la cuestión:

   Se presenta una panorámica general de la cuestión a tratar, es decir, una base teórica sobre la que se sustenta el trabajo.

4. Objetivos:

   Breve recopilación, a título general y específico, de los objetivos seguidos para este Trabajo de Fin de Grado.

5. Diseño e Implementación:

   En esta sección se detalla el trabajo realizado ofreciendo una descripción de los requisitos, arquitectura y la implementación de la aplicación.

6. Pruebas y Evaluación:

   Para cerciorarnos de que el diseño cumple con lo exigido en los requisitos, se realizarán ciertas pruebas y se mostrarán sus resultados.

7. Conclusiones:

   En esta sección se detallan las conclusiones obtenidas tras finalizar el proyecto.

8. Líneas Futuras:

   A modo de continuación del apartado anterior y con fin de que este proyecto sirva como base a futuras extensiones que se le puedan realizar, se detallan posibles mejoras sobre el mismo.

2 Gestión y Organización del proyecto

"La mejor estructura no garantizará los resultados ni el rendimiento. Pero la estructura equivocada es una garantía de fracaso".

Peter Drucker

Con el fin de alcanzar los objetivos del cliente designados para este proyecto, no basta con limitarse únicamente a la fase de implementación para conseguirlo. El desarrollo del software es un concepto más amplio que requiere de ciertas metodologías de trabajo, de un ciclo de vida del software, y de otras herramientas o mecanismos que se detallarán a lo largo del capítulo, aspirando siempre a un software de calidad.

Figura 2. Proceso de desarrollo de software

Entendiendo por software de calidad, un software, que además de cumplir con las necesidades del usuario, sea mantenible (pueda evolucionar con cambios de las necesidades), sea confiable (fiable, seguro y no cause daños económicos/físicos en caso de falla), eficiente (aproveche lo máximo posible los recursos que disponga) y usable (fácil de utilizar).

Esto último sería ciertamente así, si se pudiesen aislar el resto de variables que afectan en la realización de un proyecto y obviarlas, cosa que dista de la realidad. Variables como el tiempo o el coste del proyecto, aun pudiendo estar fijas (*scrum*, metodologías ágiles) o ser variables, se han de seguir teniendo en cuenta, ya que realizar un software de calidad requiere de un marco concreto en el que realizarlo, así como de ciertas limitaciones y usos de recursos.

Dicho esto, podría decirse que la calidad de todo proyecto software está determinada por 3 variables relacionadas entre sí, formando el triángulo que aparece en la Figura 3.

Figura 3. Triángulo de hierro (Coste, Alcance, Tiempo)

De esta forma por ejemplo, para una cierta calidad, si se redujese la fecha de entrega del proyecto, habría necesariamente que reducir el alcance y/o aumentar los recursos dedicados (si fuese posible reducir tiempo paralelizando el trabajo de tareas).

Estas variables no afectan sólo a la calidad sino también determinan la complejidad del proyecto, para el tamaño del mismo se ha aplicado el estándar que se detalla a continuación.

2.1 Estándar de Ingeniería de Software

Con el fin de definir las prácticas de software a aplicar en este proyecto se ha decidido seguir el estándar de ingeniería de software de la Agencia Espacial Europea (ESA), "PSS-05 lite", en su versión para proyectos de pequeña magnitud. Se considera de pequeño tamaño si [7]:

- Se necesitan menos de 2 años/hombre de esfuerzo para el desarrollo. ✓
- Se requiere un equipo único de desarrollo de 5 o menos personas. ✓
- La cantidad de código es inferior a 10.000 líneas (excluyendo comentarios). ✓

Además al ser un proyecto de software pequeño no crítico (no hay riesgo de pérdida de vidas, propiedades o males mayores) se intentarán seguir las siguientes estrategias siempre que sea posible:

- Documentación simplificada
- Reducir formalidad en los requisitos
- Simplificar planificación
- Combinación de requisitos software y fases de diseño arquitectónico
- Especificaciones del plan de pruebas para las pruebas de aceptación

Tras definir el estándar a emplear durante el proyecto, se prosigue con las pautas a aplicar a partir del mismo.

2.2 Metodología de desarrollo de software

Las metodologías surgen como un proceso riguroso y disciplinado de desarrollar software en respuesta a la falta de planificación, de predicción y de eficiencia de los métodos convencionales (o sin metodología). Cierto es que a pesar de haber distintos procesos de desarrollo con sus correspondientes etapas y sub-etapas, existen ciertas etapas mínimas que se cumplen siempre:

- Especificación de software (captura y análisis de requisitos)
- Diseño
- Implementación
- Validación (Pruebas unitarias/integración)
- Instalación y paso a producción
- Mantenimiento o evolución

En este proyecto, concretamente, con el fin de gestionar la complejidad del mismo se realizará un enfoque de *control predictivo*, es decir, tal y como se ejecutaría un contrato tradicional con el cliente, en el que los requisitos se establecen al comienzo y en la entrega final el producto se correspondería con dichos requisitos. Se ha decidido elegir este enfoque frente al empírico de las metodologías ágiles (*Scrum*, *eXtreme Programming*, etc.), ya que aunque este último es más adaptativo al emplear un ciclo iterativo e incremental, los requisitos del proyecto impuestos por el cliente (el tutor en este caso) son en su mayor parte invariables y están definidos todos desde el comienzo. Cabe destacar además que el cliente dispone de conocimiento técnico y de cómo desea a medio/bajo nivel establecer el alcance del proyecto. Es por esto que se dispondrá de un férreo control en el cambio de los requisitos para no desviarse de la planificación inicial. Por último comentar que el conocimiento del equipo de desarrollo en el ámbito de las metodologías ágiles es relativamente reciente y escaso, lo cual implicaría un riesgo extra en el proyecto que podría acarrear problemas y desviaciones a largo plazo.

Aun habiendo elegido dicho enfoque, se tendrá muy presente la participación del cliente mediante reuniones o seguimientos periódicos con el fin de que pueda guiar de manera regular los resultados del proyecto y más concretamente de la aplicación.

En el siguiente apartado se detalla el ciclo de vida del software en base a este enfoque secuencial citado anteriormente.

2.3 Ciclo de vida del software

Un *modelo de ciclo de vida* no es más que la determinación en el orden a seguir entre las tareas que derivan de las fases citadas en el apartado anterior (véase 2.2 Metodología de desarrollo de software), con sus correspondientes enlaces, coordinaciones y realimentaciones entre ellas.

De entre todos los ciclos de vida existentes, y más concretamente los secuenciales o lineales, se ha decidido emplear para este proyecto el *modelo en cascada realimentado*.

El modelo tradicional también denominado comúnmente modelo de desarrollo en cascada, debe su nombre a la forma secuencial en la que están establecidas cada una de las etapas del ciclo de vida. Al igual que en una cascada, dichas fases están dispuestas de tal modo que avanzar en el desarrollo es rápido y fácil, mientras que en el caso de que hubiese que corregir algo y "volver atrás" sería muy costoso. En esta analogía, a medida que se va avanzando cada vez más, volver a las primeras etapas supone un coste que se incrementa casi de forma exponencial, ya que por ejemplo, modificar el análisis implicaría seguramente modificar el diseño y el resto de tareas que lo acontecen.

Para evitar costes innecesarios se realizará una revisión tras finalizar cada etapa con el fin de comprobar y confirmar que se puede pasar a la siguiente.

Se decidió elegir el modelo en cascada frente a otros secuenciales como el modelo de desarrollo en V, ya que al estar formado el equipo de desarrollo por una única persona no se iban a poder aprovechar las ventajas de la paralelización de ambas cascadas (en el modelo en V se representan dos cascadas enfrentadas, con la codificación como vértice común). El ciclo de vida tipo *sashimi* también se descartó por esta misma razón, al existir un solapamiento entre fases y resultar incompatible con un proyecto unipersonal. Esta escasez de recursos en el equipo de desarrollo no supone un inconveniente en el modelo en cascada tradicional, ya que las distintas fases pueden ser efectuadas por una misma persona [4].

Concretamente de entre las distintas variedades del modelo en cascada, se usará, tal y como se comentó al principio, la variante *realimentada* con el fin de corregir errores pronto, pero siempre ciñéndose al desarrollo lineal.

2.3.1 Modelo de desarrollo en cascada con realimentación

Aun siendo un proyecto pequeño (rondando las 300 horas de trabajo), bien es cierto que incluso el software más nimio no se libra de estar sujeto a fallos o variaciones, y presumir que todo el proceso va a seguir la plena rigidez y no volatilidad de los requisitos y que cada etapa va a estar exenta de errores, es quizás demasiado optimista. Es por esta razón por la que el modelo en cascada "puro" rara vez se usa, optándose por alguna de sus variantes, actualmente de entre las más utilizadas [5].

La evolución es algo intrínseco en el carácter del software [6], por tanto es lógico pensar que podría ser interesante introducir realimentación entre ciertas etapas, para poder por ejemplo, modificar o añadir requisitos aun habiendo entrado en la fase de diseño. De esta forma se permite retroceder a etapas anteriores si así es requerido. En la Figura 4 se pueden apreciar las etapas de "Definición de requisitos", "Análisis y Diseño", "Implementación y Pruebas unitarias", "Integración y Pruebas de Sistema" y "Operación y Mantenimiento".

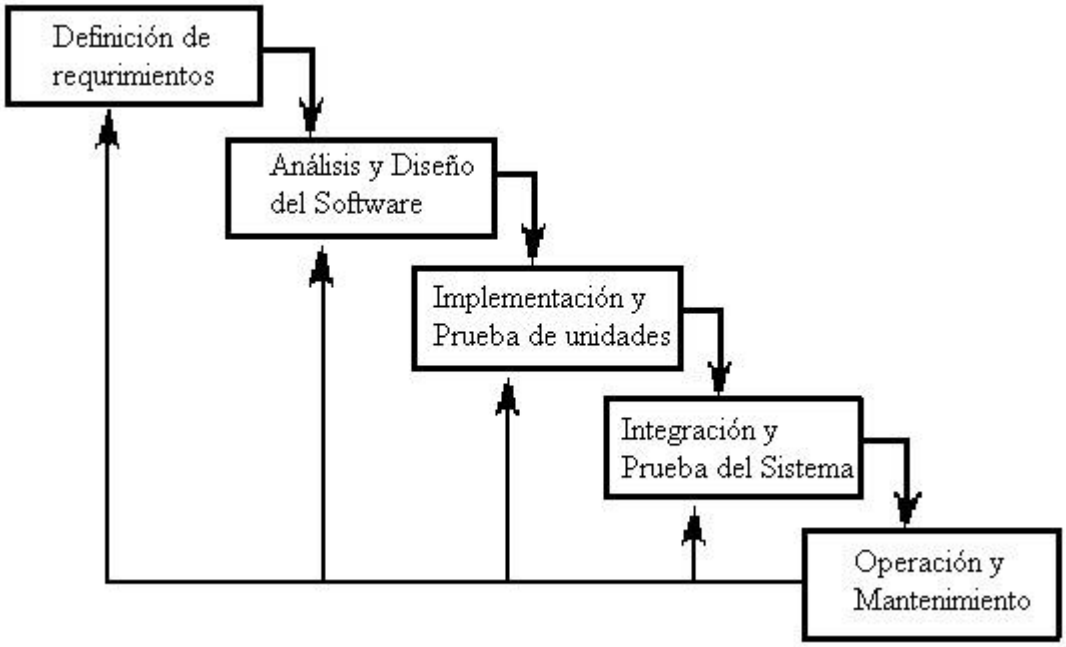


Figura 4. Modelo en Cascada Realimentado

Para un proyecto como el que nos acontece, resulta casi ideal este tipo de modelo ya que se trata de un proyecto de alta rigidez, con requisitos correctamente especificados y muy claros.

Al fin y al cabo la realimentación sirve como medio de reducción de riesgos, por ejemplo, aplicando como se ha comentado anteriormente, a la fase de análisis y a la de diseño un desarrollo iterativo:

1. Consultar al cliente/usuario.
2. Realizar diseño que se desprende del punto 1.
3. Realizar prototipo de interfaz, entrevistas, etc. y volver con ello al punto 1, para poder obtener nuevos requisitos o modificar malentendidos.

El desglose de las distintas fases sería por tanto el siguiente:

1. Análisis de requisitos

En base a lo demandado por el cliente y a las necesidades de los usuarios finales se determinan cuáles son los objetivos a cumplir y se especifican plasmados en forma de requisitos.

2. Diseño del sistema

Es en esta fase donde se realiza la transición entre lo especificado en el análisis (requisitos) y el diseño del sistema (diseño de alto nivel o diseño arquitectónico, y diseño detallado).

El diseño arquitectónico se centra en definir la estructura de la solución, identificar los módulos y relacionarlos. El diseño detallado, por su parte, es a más bajo nivel (algoritmos, definir la organización del código, etc.).

3. Implementación

Esta fase suele acaparar la mayor parte del tiempo, traduciendo a código fuente el diseño detallado previamente descrito. Como resultado se construirían los módulos y las unidades software.

4. Pruebas

Tras la implementación, se realizarían pruebas unitarias y se integrarían todas las unidades probándose en conjunto, y verificando como es obvio que cumpla con lo demandado por el cliente.

5. Mantenimiento

Una vez ya se ha entregado el producto final al cliente, se comienza esta última fase que suele ser generalmente más larga (se acuerda con el cliente), en la que se realizan correcciones de errores descubiertos, posibles mejoras sobre la implementación, nuevos requisitos, etc.

2.4 Requisitos de usuario y de software

Este apartado se centra en un área primordial para el proyecto y es el de definir qué es lo que se desea producir. Las necesidades del cliente o de los usuarios deberán de quedar reflejadas en forma de *requisitos*, sin ambigüedades, consistentes y descritos con claridad.

Tal y como anuncia el título de esta subsección y según el estándar en el que se basa este proyecto, se pueden distinguir dos tipos de requisitos: Requisitos de *usuario* y requisitos de *software*. Ambos tipos cuentan con los mismos campos a la hora de ser especificados tal y como se muestra en la Tabla 1.

ID: RS-NF01					
Nombre:	Nombre del requisito				
Tipo requisito:	Manejo de errores				
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional				
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	Descripción del requisito				

Tabla 1. Ejemplo de especificación de requisito

Los identificadores posibles son los siguientes:

- **RU-C** + nº de requisito: Requisito de usuario de capacidad
- **RU-R** + nº de requisito: Requisito de usuario de restricción
- **RS-F** + nº de requisito: Requisito de software funcional
- **RS-NF** + nº de requisito: Requisito de software no funcional

Es esencial además verificar la trazabilidad de los requisitos tanto "hacia atrás" como "hacia delante", es decir, todo RU debe ser desarrollado por al menos un RS (*hacia atrás*) y los requisitos han de corresponderse con cierta o ciertas partes del código de la aplicación (*hacia delante*). Estas comprobaciones se llevarán a cabo mediante lo que comúnmente se conoce como "Matrices de Trazabilidad" [8].

A continuación cada uno de los tipos de requisitos más en detalle:

2.4.1 Requisitos de usuario

Se realiza una captura de requisitos con el planteamiento del problema como objetivo. Dichos requisitos son definidos por el *usuario/cliente* y el analista deberá tomar nota de ellos. En este tipo de requisitos se hace especial énfasis en características de los usuarios y en la perspectiva del producto sin entrar nunca en un nivel de detalle demasiado técnico. Por tanto, en base al tipo de requisito que declare el usuario, se distinguirán entre requisitos de *capacidad* (qué es lo que debe de hacer la aplicación) y requisitos de *restricción* (limitaciones de la aplicación; cómo debe hacerlo).

2.4.2 Requisitos de software

Partiendo de los requisitos de usuario, el *analista* realizará un refinamiento del problema y en este caso, el desarrollador hará de audiencia con el fin de posteriormente implementar el software en base a estas necesidades. Esta vez definidos a un nivel más técnico de detalle, se hará uso del conocimiento de expertos, de métodos formales, etc. Dichos requisitos se clasifican principalmente en requisitos *funcionales* (misma función que los de *capacidad* definidos anteriormente, pero a un nivel más bajo) y requisitos *no funcionales* (misma función que los de *restricción* definidos anteriormente, pero a un nivel más bajo).

Se ha creído conveniente desglosar a su vez los requisitos *no funcionales* siguiendo el estándar de la ESA según los siguientes tipos: [1] [2]

- Consumo de recursos (memoria, capacidad de tráfico, etc.)
- Rendimiento (velocidad, tiempo de respuesta, etc.)
- Fiabilidad y disponibilidad (cuantificación de fallos permitidos)
- Manejo de errores (errores de entorno, errores internos)
- Requisitos de interfaz (comunicación con usuarios, con otras aplicaciones)
- Restricciones (exactitud, lenguajes y plataformas, arquitectura, estándares, etc.)
- Seguridad (seguridad del sistema, encriptación, etc.)

[1] *Es posible que para esta aplicación en concreto no haya requisitos para todos y cada uno de los distintos tipos de no funcionales.*
[2] *Dichos requisitos se especificarán en el apartado* 5.1.1 Requisitos de la aplicación.

2.5 Diseño arquitectónico y detallado

Tras realizar el análisis de requisitos y teniendo en cuenta el ciclo de vida empleado para este proyecto, el siguiente paso sería modelar dicho conocimiento en forma de diseño del software. El conocimiento bien representado ayuda a hacerse las preguntas adecuadas: ¿Por qué no se puede...? ¿Qué falta aquí? ¿Qué pasaría si...?

En cuanto al diseño, se podría hablar de dos fases con un distinto nivel de especificación: el *diseño arquitectónico* y el *diseño detallado*. El diseño arquitectónico describe a un alto nivel cómo se descomponen los distintos componentes en el software y cómo están organizados, pero siempre sin entrar en detalles de implementación. El diseño detallado, en cambio, describe el comportamiento de los componentes citados anteriormente de una forma específica y a más bajo nivel.

En general en todo diseño de un sistema se han de tener en cuenta al menos dos conceptos primordiales: la *cohesión* y el *acoplamiento* [9]:

- Acoplamiento: El acoplamiento hace referencia al grado de dependencia entre dos unidades software (métodos, clases, librerías, aplicaciones, componentes, etc.), es decir, en qué medida puede una hacer su trabajo sin depender de la otra. El objetivo será por tanto obtener un acoplamiento lo más bajo que sea posible, para que a la hora de implementar exista una mejora en la detección y corrección de futuros errores, y se facilite el mantenimiento y la reutilización en mayor medida.

- Cohesión: La cohesión hace referencia a la forma en la que agrupamos las unidades software en una unidad mayor. De esta forma se mide en menor o mayor medida la identidad del comportamiento de un componente dado (o unidad mayor). Para poder evitar que un componente realice más operaciones o responsabilidades de las que debiera, la cohesión debe ser lo más alta posible, es decir, cuanto más cohesionados estén los elementos que se agrupan, mejor. En este proyecto se llevará a cabo una *cohesión funcional*, de forma que las unidades software se agruparán en función de la finalidad que acometan.

Para lograr una buena descomposición modular acertando de antemano y que se cumplan los criterios mencionados anteriormente, lo más conveniente es emplear estilos arquitectónicos. De entre todos los estilos arquitectónicos existentes (arquitecturas de máquinas virtuales, tuberías y filtros, arquitectura en capas, *peer-to-peer*, etc.), se va a emplear el Modelo Vista Controlador para este proyecto[3], ya que es fácilmente extensible, simple, eficiente, y permite separar con facilidad los componentes del sistema [10].

De la misma forma que en los requisitos se verificaba su trazabilidad, (requisitos de usuario en base a los requisitos de software), todo componente de dicha arquitectura debe provenir de alguno de los requisitos de software con el fin de satisfacer la verificación. Tras verificar de este modo que el diseño cumple con las exigencias del usuario podría darse paso a la etapa de codificación, pero no antes.

[3] *Dicho diseño se tratará con más detalle en el apartado* 5.2.1 Arquitectura de la aplicación

2.6 Implementación del SW

Como es lógico, tras el análisis y el diseño, ya se dispone de todo el conocimiento necesario para dar paso a la implementación del programa.[4]

Esta es la etapa que cobra más importancia de cara al usuario final, ya que materializa el esfuerzo previo realizado en algo tangible por el cliente. Es además una de las etapas más largas (horas de trabajo) en llevar a cabo, y a pesar de contar con una cantidad menor de documentación escrita, abordará temas necesarios como las decisiones de implementación, problemas en la codificación, lenguaje de programación empleado, etc.

2.7 Evaluación de los resultados

Una vez realizada la codificación y con el software listo tras ser implementado, es necesario comprobar que dicha aplicación funciona como es debido y que se corresponde con los objetivos que se establecieron al comienzo del proyecto.

Es bien sabido que obtener un programa perfecto y libre de fallos es algo casi utópico e inaccesible en la mayoría de los casos, a no ser que la complejidad del mismo sea realmente reducida [11]. Existen autores incluso que afirman que los errores en el software son inherentes a su naturaleza debido a su entidad no física [12]. Además hay que tener en cuenta que la depuración de sistemas software sigue la ley de rendimiento decreciente, es decir, a medida que avanza el proceso de búsqueda de errores, llega un punto temprano a partir del cual el coste de detección supera por mucho el beneficio obtenido.

Dicho esto, el cometido de esta fase no será tanto la eliminación completa de errores, sino la depuración de faltas frecuentes que puedan entorpecer el sistema, ya que como se comentó anteriormente, la corrección de faltas infrecuentes no afecta a la fiabilidad percibida por la aplicación.

Para ello se empleará una serie de pruebas del sistema, a modo de pequeñas ejecuciones del software con el fin de explorar todas las posibles vías en busca de errores o inconsistencias (de datos o con la arquitectura en sí).[5]

[4] *Lo referente a la implementación del software se verá con más detalle en el apartado* 5.3 Implementación.
[5] *La evaluación se verá con más detalle en el apartado* 6 Pruebas y evaluación.

3 Estado de la cuestión

"La inspiración existe, pero tiene que encontrarte trabajando".

Pablo Picasso

Este capítulo sirve como base teórica sobre la que se sustenta este Proyecto de Fin de Grado, y en el que se ofrece una visión general del área en el que se enmarca el mismo.

Se comienza hablando sobre el sistema operativo a emplear, desde un punto de vista histórico, su arquitectura y funcionamiento interno, características, diferencias entre versiones y comparación con otros SOs (tanto en características como en mercado). Seguidamente se comenta el protocolo de comunicación empleado, *publish/subscribe* más concretamente la tecnología MQTT, y se detalla la arquitectura referente al *backend* ya establecida. Se prosigue con aplicaciones existentes similares a la que se persigue en este proyecto y finalmente se comentan las herramientas empleadas para la realización del *frontend*.

3.1 Android

Basándose en Linux surgió Android como un sistema operativo enfocado a los dispositivos móviles, tales como *smartphones*, y más recientemente *tablets* o *Google TV*. Su desarrollo está liderado por la *Open Handset Alliance* la cual fue fundada por *Google* para promover y desarrollar estándares abiertos para ser usados en dispositivos móviles.

A lo largo de esta sección se entrará en más detalle en este sistema operativo, elegido para implementar la aplicación de este TFG. Dicha elección es en base a lo expuesto en este capítulo (concretamente a partir de la sección 3.1.4) así como a las preferencias de la arquitectura ya implantada en el *backend* (véase sección 3.2.2.2).

3.1.1 Historia de Android

El largo viaje del sistema operativo Android, a tal y como lo conocemos en la actualidad, comenzó en una pequeña empresa fundada en 2003 por Andy Rubin, Rich Miner, Nick Sears y Chris White, entre otros. Todos ellos co-fundadores o liderando el desarrollo en empresas como Danger, Wildfire o T-Mobile, que posteriormente se fueron a trabajar a Google. Esto ocurrió cuando en julio de 2005 Google adquirió esa pequeña compañía con la mentalidad de poder entrar en el mercado de la telefonía móvil. Dicha empresa se hacía llamar *Android Inc.* y fue fundada en Palo Alto (California).

Ya una vez contratados por Google, el equipo que lideraba Rubin desarrolló una plataforma basada en el kernel de Linux para dispositivos móviles, con la promesa de ofrecer un sistema flexible y actualizable. En ese momento Google ya había comenzado a mantener relaciones entre fabricantes hardware y software, abriendo su posible cooperación a operadores móviles.

En 2006 se incrementó la especulación de que el sistema Android entraría en el mercado y en 2007 "*InformationWeek*" difundió un estudio reportando que Google había solicitado diverstas patentes dentro de la telefonía móvil [23]. Ese mismo año se fundó el consorcio *Open Handset Alliance*, y se estrenó la primera plataforma Android construida sobre el kernel 2.6 de Linux, cerrando así todo tipo de especulaciones.

Fue finalmente en septiembre de 2008 cuando se lanzó la versión 1.0 de Android (también conocida como "A") integrada en el teléfono móvil "*HTC Dream*".

Por último y a título de curiosidad, tanto el nombre del sistema operativo (Android) como alguno de los smartphones de *Google (Nexus One)*, hacen referencia a la novela "¿Sueñan los androides con ovejas eléctricas?" de *Philip K. Dick* y a su posterior versión adaptada al cine de *Blade Runner* [24].

3.1.2 Características y versiones

3.1.2.1 Características generales

Cada distinta versión de Android que es lanzada dispone de nuevas mejoras y características específicas, pero como base, el sistema operativo fue diseñado para soportar las siguientes:

- Diseño de dispositivo:
  La plataforma es adaptable a pantallas más grandes, a VGA, a biblioteca de gráficos 2D y 3D, y en general al diseño de móviles tradicionales.

- Almacenamiento:
  Emplea la base de datos SQLite por ser ligera y con propósitos de almacenamiento de datos.

- Conectividad:
  Dentro del ámbito de la conectividad, Android soporta las siguientes tecnologías: *EV-DO, CDMA, GSM/EDGE, IDEN, UMTS, LTE, Bluetooth, Wi-Fi, HSDPA, HSPA+* y *WiMAX*.

- Mensajería:
  Soporta *SMS* y *MMS* como formas tradicionales de mensajería de texto, así como *C2DM* (*Android Cloud to Device Messaging*), como parte del servicio que ofrece Android de *Push Messaging*.

- Navegador web:
  El navegador que se incluye por defecto en Android está basado en *WebKit*, junto con el motor *JavaScript V8* de *Google Chrome*. En versiones de Android recientes se incluye *Google Chrome* por defecto.

- Soporte de Java:
  No existe una máquina virtual de Java en el dispositivo a pesar de que la mayoría de las aplicaciones estén escritas en Java. El soporte para *J2ME* puede ser añadido mediante aplicaciones de terceros (*J2ME MIDP Runner* [14]).

- Soporte multimedia:
  Dentro del ámbito del multimedia Android soporta los siguientes formatos: *WebM, H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF* y *BMP* [15].

- Soporte para streaming:
  Android soporta el Streaming *RTP/RTSP*, la descarga progresiva de *HTML* mediante el tag <*video*> de *HTML5* y el *Adobe Flash Streaming* (mediante *Adobe Flash Player*). Se planea el soporte de *Microsoft Smooth Streaming* con el plugin de *Silverlight* para Android.

- Soporte para hardware adicional:
  Soporte para pantallas táctiles, *GPS*, acelerómetros, cámaras de fotos y de vídeo, giroscopios, magnetómetros, gamepad, termómetro, sensores de proximidad y de presión, y aceleración por *GPU* (2D y 3D).

- Entorno de desarrollo:
  El entorno integrado es *Eclipse* empleando un plugin de "Herramientas de Desarrollo de Android", e incluye herramientas para depuración de memoria, un emulador de dispositivos y análisis de rendimiento del software.

- Google Play:
  Google Play sustituye al anterior *Android Market* con aplicaciones tanto gratuitas como de pago y que pueden ser obtenidas sin necesidad de un PC.

- Multi-táctil:
  Android ofrece soporte multi-táctil de forma nativa para pantallas capacitivas.

- Bluetooth:
  Ofrece soporte para *A2DF* y *AVRCP*, así como envío de archivos (*OPP*), exploración del directorio telefónico, marcado por voz y envío de contactos entre teléfonos.

- Videollamada:
  A partir de *HoneyComb* Android soporta videollamadas a través de su aplicación de *Google Talk*.

- Multitarea:
  Se dispone de multitarea real de aplicaciones, recibiendo ciclos de reloj las aplicaciones que no se estén ejecutando en primer plano.

- Características basadas en voz:
  Soporta la búsqueda en Google a través de voz.

- Tethering:
  Permite al dispositivo ser usado como punto de acceso inalámbrico (y alámbrico), aunque para permitir a un PC usar la conexión 3G del dispositivo, podría ser necesaria la instalación de software adicional [16].

3.1.2.2 Características en función de versión

Google se ha dedicado desde 2008 a mejorar a un ritmo acelerado la versión del sistema operativo Android por lo que en esta sección se revisarán las características principales de cada una de ellas.

Cabe destacar que Android fue creado en un principio para emplearse en *smartphones*, pero con la entrada de las Tablets en el mercado, Google se vio obligado a hacer distinción entre las versiones. Por ello las versiones para smartphones engloban desde la 1.0 a la 2.XX, las Tablets tienen asignadas el abanico de la 3.XX, y recientemente con la última generación, la 4.XX se unifica el diseño para poder emplearse en cualquier dispositivo.

A continuación las distintas versiones:

- **1.0**: Fue liberada el 23 de Septiembre de 2008 [17]
- **1.1**: Fue liberada el 9 de Febrero de 2009 [18]

(A partir de aquí las versiones oficiales, que curiosamente están ordenadas por orden alfabético y con nombre de postes o de comida dulce [19]).

- **1.5** – Android "**Cupcake**":
  (Basado en el Kernel de Linux 2.6.27)
  - Lanzamiento: 30 de Abril de 2009
  - Funciones: Capacidad de subir videos a YouTube e imágenes a Picasa de forma directa, soporte para Bluetooth (*A2DP* y *AVRCP*), teclado con predicción de texto, transiciones de pantallas animadas, grabar y reproducir videos, y nuevos *widgets*.
  - Notas: A pesar de que existieran versiones previas de Android, esta marca el inicio del S.O.

Figura 5. Android Cupcake logo

- **1.6** – Android "**Donut**":
  (Basado en el Kernel de Linux 2.6.29)
  - Lanzamiento: 15 de Septiembre de 2009
  - Funciones: Se mejora la experiencia en el Android Market, soporte para *CDMA/EVDO*, 802.1x, *VPN* y *WVGA*, mejora velocidad en aplicaciones de búsqueda y cámara, búsqueda por voz y navegación gratuita *turn-by-turn* de *Google*.
  - Notas: Versión barata que no encarecía el coste de los dispositivos, e inicia de ya de forma importante el auge en la comunidad de desarrolladores.

Figura 6. Android Donut logo

- **2.0 / 2.1** – Android "**Eclair**":
  (Basado en el Kernel de Linux 2.6.29)
  - Lanzamiento: 3 de Diciembre de 2009 y 12 de Enero de 2010 respectivamente
  - Funciones: Velocidad hardware optimizada, soporte para *HTML5*, para *Microsoft Exchange*, más resoluciones y para el flash de la cámara. *Bluetooth* 2.1, mejoras en *Google Maps 3.1.2* y en teclado virtual, nuevas listas de contactos, IU mejorada, fondos de pantalla animados y zoom digital.
  - Notas: Comienza a demostrar su estabilidad y robustez como S.O. para smartphones.

Figura 7. Android Eclair logo

- **2.2** – Android "**Froyo**":
  (Basado en el Kernel de Linux 2.6.32)
  - Lanzamiento: 20 de Mayo de 2010
  - Funciones: Se produce una optimización general sobre memoria, rendimiento y velocidad de aplicaciones; se integra el motor *JavaScript V8* de *Google Chrome*; *Wi-Fi hotspot* y *tethering* por *USB*; permite desactivar el tráfico de datos; marcación por voz; soportes varios (contraseñas, carga de archivos en el Browser, instalación de aplicación en memoria expandible, *Adobe Flash 10.1*, pantallas con alto número de "puntos por pulgada", etc.).
  - Notas: Los usuarios aceptan el sistema operativo de forma excelente y comienza a asentarse a nivel mundial.

Figura 8. Android Froyo logo

- **2.3** – Android "**Gingerbread**":
  (Basado en el Kernel de Linux 2.6.35.7)
  - Lanzamiento: 6 de Diciembre de 2010
  - Funciones: Soporte para pagos mediante *NFC*, nativo para *VoIP SIP*, múltiples cámaras, para *WebM/VP8* y *AAC*, para más sensores (giroscopio, barómetro); mejoras en batería y administrador de tareas, en entrada de datos (para desarrolladores de juegos), en video online y en el teclado virtual; cortar/copiar/pegar a lo largo del sistema; sistema de archivos *ext4*.
  - Notas: Ofrece más posibilidades a usuarios que estén totalmente conectados.

Figura 9. Android Gingerbread logo

- **3.0 / 3.1 / 3.2** – Android "**Honey Comb**":
  - Lanzamiento: 22 de Febrero de 2011
  - Funciones: Se realiza una mejora en el sistema multitarea (multitasking), escritorio en 3D, mejoras en el navegador web, soporte mediante *Google Talk* para videochat, mejora el soporte para *Wi-Fi*, añade soporte para una gran variedad de periféricos y los widgets pueden ser redimensionados (sin la limitación del número de cuadros).
  - Notas: Versión optimizada para tablets, la versión 2.3 se mantiene para smartphones.

Figura 10. Android Honey Comb logo

- **4.0** – Android "**Icecream Sandwich**":
  - Lanzamiento: 19 de Octubre de 2011
  - Funciones: Nueva interfaz y nueva fuente, opción de emplear los botones virtuales, aceleración por hardware, mejora en la multitarea, gestor de tráfico de datos, mejora en el corrector de texto, modificación de notificaciones, mejora de la aplicación de la cámara, *Android Beam* (para compartir contenido entre teléfonos), reconocimiento facial y por voz, soporte nativo para *MKV* y *Stylus*, y nuevo framework para aplicaciones.
  - Notas: Primera versión que unifica tanto el diseño para tablets como para smartphones.

Figura 11. Android Icecream Sandwich logo

- **4.1** – Android "**Jelly Bean**":
  - Lanzamiento: 28 de Junio de 2012
  - Funciones: Empleo de *Project Butter* para la mejora de la velocidad, optimizando la interfaz para ofrecer mayor rapidez en los efectos y no existan "tirones". Modificación de la barra de notificaciones (más sencilla y limpia) y la introducción de gestos. Nuevo sistema de predicción de texto, dictado de voz *offline*, empleo de voz tanto para búsquedas como para el sistema en sí y *Google Now* [20]. *Google Chrome* como navegador por defecto, fin al soporte de *Flash Player* y cifrado de aplicaciones.
  - Notas: Última versión hasta la fecha del sistema operativo de Google para smartphones/tablets.

Figura 12. Android Jelly Bean logo

3.1.2.3 Comparativa versiones

En la Figura 13 se puede observar una gráfica de la cuota de mercado de las distintas versiones del sistema operativo Android a fecha del 2 de Julio de 2012 tras un periodo de recogida de datos de 14 días [21].

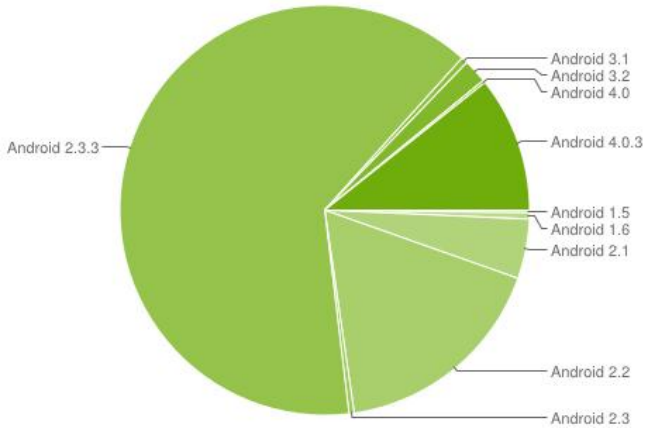


Figura 13. Versiones Android – Cuota de mercado 2012

Versión	Fecha de Lanzamiento	Nivel API	Distribución
4.1.x Jelly Bean	28 de Junio, 2012	16	Aún desconocida
4.0.x Ice Cream Sandwich	19 de Octubre, 2011	14-15	10.9%
3.x.x Honeycomb	22 de Febrero, 2011	11-13	2.1%
2.3.x Gingerbread	6 de Diciembre, 2010	9-10	64%
2.2 Froyo	20 de Mayo, 2010	8	17.3%
2.0, 2.1 Eclair	26 de Octubre, 2009	7	4.7%
1.6 Donut	15 de Septiembre, 2009	4	0.5%
1.5 Cupcake	30 de Abril, 2009	3	0.2%

Tabla 2. Versiones Android – Cuota de mercado

Basándose en la gráfica anteriormente expuesta, la Tabla 2 indica en más detalle como la versión 2.3 (Gingerbread) es la más extendida, ya que desde principios de 2011 se empezó a incluir en todos los teléfonos de gama media-alta, y es la última antes de que se unificara el diseño de smartphones/tablets en la 4.0. Le siguen de lejos la 2.2 (con un 17%) y la 4.0 (con un 11%). La 2.2 está en el segundo puesto ya que gozaba a mediados de 2011 con casi un 50% de la cuota de mercado (véase Figura 14), pero contaba con fallos de seguridad importantes resueltos en la 2.3, por lo que su cuota se vio mermada. La 4.0 comenzará a extenderse junto con la 4.1 en los próximos meses, ya que tanto tablets como smartphones hacen uso de ellas, y son las versiones más recientes del S.O. que comenzarán a integrarse en los dispositivos más recientes.

Para comparar como ha sido la evolución respecto al año anterior (finales de Abril de 2011), véase la Figura 14.

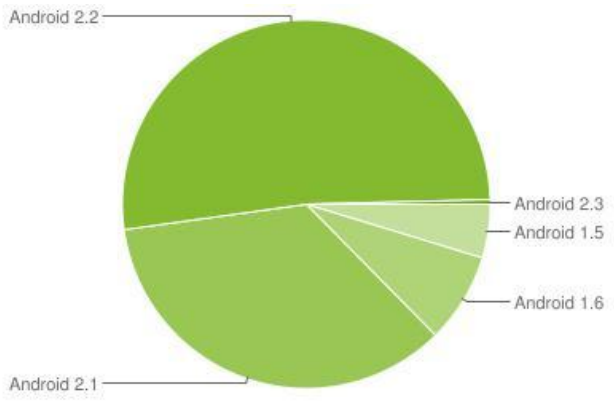


Figura 14. Versiones Android – Cuota de mercado 2011

Para poder apreciar dicho periodo de forma temporal, la gráfica de la Figura 15 muestra el número de dispositivos Android que accedieron a *Google Play* desde el 1 de Enero de 2012 hasta el 2 de Julio del mismo año.

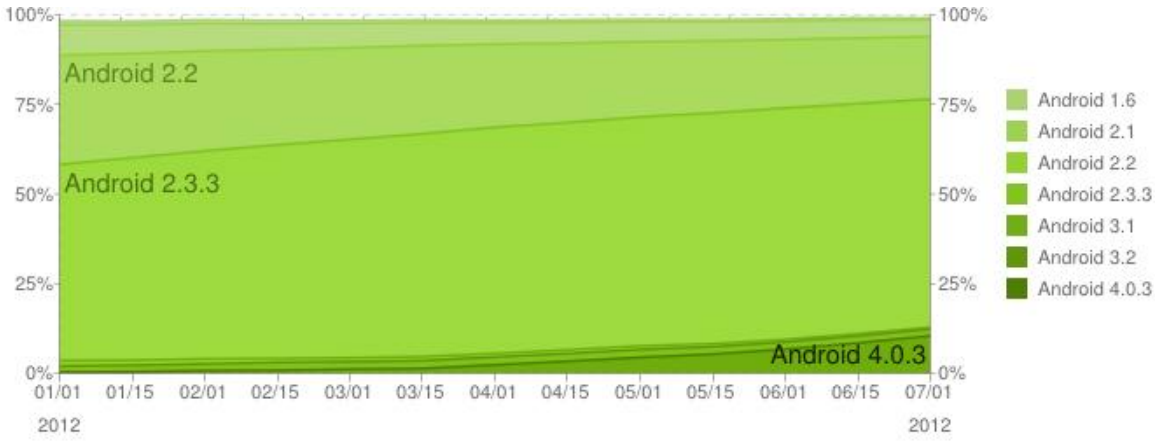


Figura 15. Versiones Android – Distribución histórica Julio 2012

Tal y como se comentó anteriormente, mientras que la versión 2.2 está "estancada" y probablemente comience a decrecer en un breve periodo de tiempo, la 2.3.3 y sobre todo la versión 4.x siguen una curva creciente, es decir, cada vez más dispositivos y por tanto más usuarios gozarán de estas nuevas versiones.

Esta gráfica también ofrece la valiosa información del número de dispositivos compatibles en base a la versión de una determinada aplicación. El formato en el que está dibujada la gráfica, establece las versiones activas más antiguas en la parte más alta, indicando el porcentaje de dispositivos compatibles para una cierta versión de Android, es decir, si se emplease la versión que está por encima de la 2.2 (1.6 o 2.1), la aplicación a desarrollar sería compatible con el 100% de los dispositivos. Si por el contrario se desarrollase para la 4.0.3, actualmente, se dispondría de un escaso 15% de compatibilidad. Esto se debe a que Android es 100% compatible "hacia delante" (*forward compatible*) [22], es decir, apps escritas correctamente para viejas versiones, funcionan en las más recientes.

3.1.3 Arquitectura

A continuación se muestran los principales componentes del sistema operativo Android en forma de diagrama:

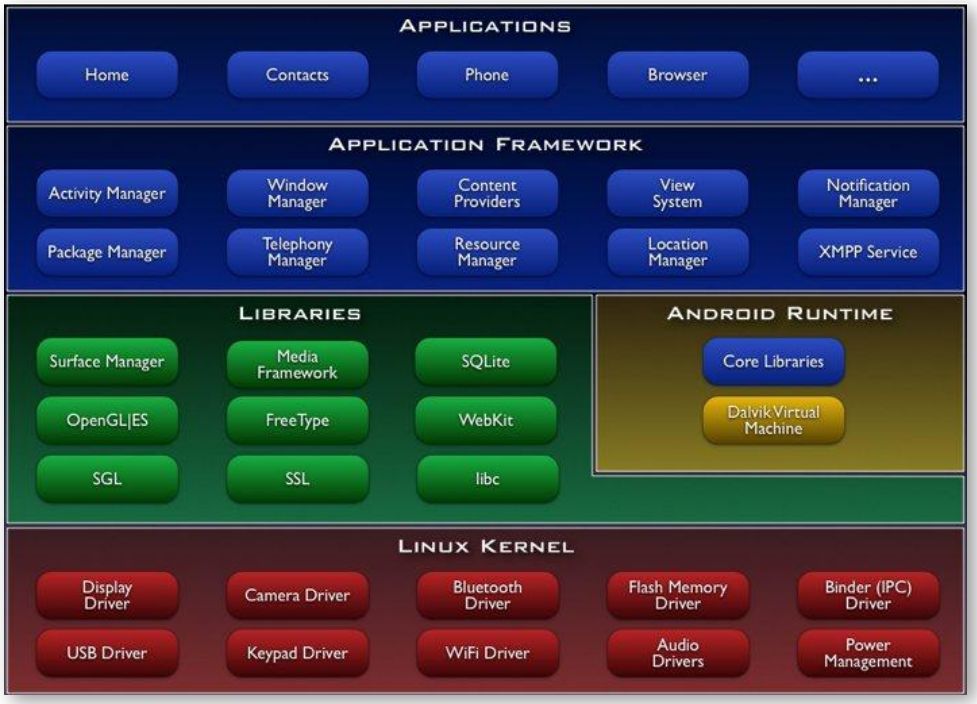


Figura 16. Diagrama Arquitectura Android

Tal y como se puede apreciar en la Figura 16, la arquitectura de Android se basa en: un kernel basado en el kernel de Linux, con middleware, librerías y APIs escritas en lenguaje C, y aplicaciones que se ejecutan en un *framework* de aplicaciones que incluye librerías compatibles con Java. Android usa a su vez la máquina virtual de Dalvik en lugar de la máquina virtual de Java con el fin de optimizar lo máximo posible los recursos existentes, en un entorno en el cual el procesador, la memoria y el almacenamiento son escasos [13].

El desglose por capas facilita la separación de tareas e implica que cada una de ellas dependa de la capa inferior para poder realizar su cometido. A este tipo de arquitectura se le denomina comúnmente *pila*. Cada una de las distintas capas (de abajo hacia arriba) en más detalle:

- Kernel de Linux:
  Se comentó anteriormente que el kernel del SO Android está basado en el kernel de Linux, más concretamente en la versión 2.6. Dicha capa actúa como intermediaria entre la pila de software y el hardware, y aporta los servicios base del sistema como el modelo de controladores, la gestión de procesos y de memoria, la pila de red o la seguridad.
  El desarrollador no emplea de forma directa esta capa, sino que accede a las librerías que se encuentran disponibles en las capas superiores.

- Librerías:
  Esta capa la componen las bibliotecas nativas de Android, las cuales están escritas en C o C++, generalmente creadas por el fabricante del teléfono el cual se encarga de instalarlas en el mismo antes de lanzarlo al mercado.
  Dichas librerías cumplen la tarea de proporcionar funcionalidades que se repiten frecuentemente como, por ejemplo, el motor gráfico (OpenGL), el cifrado de las comunicaciones (SSL) o la renderización de fuentes (FreeType).

- Runtime de Android:
  El entorno de ejecución del sistema operativo Android está formado por librerías por lo que no se podría considerar una capa en sí mismo. Dichas librerías aglutinan las funcionalidades más comunes de Java y algunas específicas del SO.

  Las distintas aplicaciones son codificadas en Java y se compilan en un formato específico (.dex) para que la máquina virtual Dalvik las ejecute. Esto tiene la ventaja de que una vez compiladas, podrán distribuirse y ejecutarse con total garantía sólo disponiendo de la versión mínima de Android que necesite la aplicación.

  La máquina virtual empleada (Dalvik) no es compatible con el Java bytecode. Java se emplea únicamente como lenguaje de programación, pero los ficheros ejecutables que genera el Android SDK están en extensión .dex específicamente para Dalvik.

- Marco de trabajo de Aplicaciones:
  Las aplicaciones emplean directamente las clases y servicios que se encuentran en esta capa para realizar sus funciones. Se diseñó de esta forma para hacer más simple la reutilización de componentes empleadas por las distintas aplicaciones. La mayoría de esos componentes que se aprecian en el diagrama, son librerías Java que acceden a través de Dalvik a recursos de capas inferiores. Entre algunos de esos componentes se encuentran las Vistas (permiten construir botones, cuadros de texto, etc.) o el Multimedia (reproducir video, audio, etc.).

- Aplicaciones:
  Por último la capa que aglomera todas las aplicaciones instaladas en el dispositivo (nativas, administradas, pre-instaladas, con o sin interfaz de usuario, etc.). Además de las aplicaciones que trae "de serie" el teléfono, el propio usuario podrá instalar las que desee mediante *Google Play* o también conocido como el market de apps de Android. Este es uno de los potenciales de Android frente a otra competencia y es que existe un control total del software que se ejecuta en el teléfono por parte del usuario.

3.1.4 Comparativa SOs

Actualmente con la existencia de los smartphones en el mercado, es habitual incorporar la efectividad de los sistemas operativos a los dispositivos móviles, ya que estos últimos precisan cada vez de más funcionalidades y de mayor cantidad de datos a procesar. Suelen ser sistemas operativos más simples que los convencionales, orientados a realizar otras funciones (como las de telefonía) y que emplean mucho *middleware* para cumplir su cometido.

A continuación se compararán algunos de los SOs más destacables, comentando algunas de sus características. Dichos sistemas operativos son: Android, Windows Phone, BlackBerry OS y iOS. Obviamente existen otros sistemas operativos conocidos como Symbian OS, Windows Mobile, WebOS o Bada, pero se han descartado porque o bien están en drástico decrecimiento (Symbian OS), porque son sistema nuevos que no llegaron a cumplir las expectativas (Bada [25]), o porque no juegan una baza demasiado importante (WebOS solo se encuentra en dispositivos Palm y en algunos HP).

En concreto se compararán las versiones de los sistemas operativos mencionados en la Figura 17 tanto en cuanto a sus características como a cuota de mercado, aplicaciones disponibles, etc.:

Figura 17. iPhone iOS vs Android vs Blackberry OS vs Windows Phone

3.1.4.1 Características

En este apartado se recorrerán las características principales de los sistemas operativos móviles arriba mencionando, comentando tanto sus pros como sus contras: [27][28][29]

Introducción

- **iOS**: El primer iPhone nació en Junio de 2007 por Apple. Lo que en 2010 se conocía como "iPhone OS", cambió su nombre a "iOS" para incorporar el iPad, iPod Touch y el Apple TV. El sistema operativo es excelente en casi todos los sentidos, pero tiene la pega principal de que es algo caro y que para remover las restricciones del iPhone se requiere tiempo, esfuerzo y conocimiento técnico para hacer un *Jailbreak*.

- **Android**: Android apareció por primera vez en un teléfono en Octubre de 2008. Un proverbio africano afirma que "se necesita todo un pueblo para educar a un niño", y Android es un buen ejemplo. No sólo está apadrinado por *Google* sino también por los miembros de la *Open Handset Alliance*, muchos de los cuales pertenecen a las compañías más grandes del mundo (*Intel*, *Motorola*, *Samsung*, *LG*, *HTC*, *Dell*, *Sony*, etc.). Este sistema operativo tiene la ventaja de que es más asequible al bolsillo, muy personalizable y más integrado con servicios online.

- **BlackBerry**: El más veterano de los cuatro, el *BlackBerry* OS proviene de *Research in Motion*. Cuando debutaron en 1999, los dispositivos de *BlackBerry* eran simples máquinas para mandar e-mails, y eso era todo. Ahora ejecutándose en la versión séptima del sistema operativo se ha incrementado sustancialmente el número de funcionalidades de las que dispone. Es fantástico para ámbito empresarial, pero aún le falta mucho por pulir para hacerse interesar por el usuario medio. Sigue siendo el mejor para el mailing, contactos, calendario y para sincronizarse con *Microsoft Outlook*.

- **Windows Phone**: El novato. El sucesor de *Windows Mobile* OS (y usurpador de *Symbian*), *Windows Phone 7.5* es el recién nacido de *Microsoft* y apareció por primera vez en Noviembre de 2010. *Windows Phone* fue un cambio importante en el enfoque de pasar de un mundo empresarial a uno consumidor, y *Microsoft* por su parte abandonó el soporte de las características de muchos negocios para poder llevar esta primera iteración a cabo. La adopción ha sido lenta, pero tras asociarse con *Nokia* [26], muchos analistas predicen un crecimiento acelerado. Es un sistema operativo elegante, fácil de usar y potente, sin embargo el hecho de tener baja cuota de mercado significa que tiende a ser menor el apoyo por parte de apps y websites.

Generalidades

- **Apps**: En cuanto a las apps, iOS y Android están en cabeza con una enorme selección y variedad. iOS impone algunas limitaciones (bloquea ciertas apps como *Swype*, *Grooveshark* o *Google Voice*), pero disfruta de más calidad de media que Android. Android por su parte no establece barreras y por tanto aparecen aplicaciones con peor calidad. BlackBerry tiene una selección muy limitada y de baja calidad. Por último Windows Phone dispone de una nueva librería de apps que aunque aun es pequeña está creciendo rápidamente.

- **Usabilidad & Diseño**: iOS dispone de una fantástica usabilidad, y de una interfaz elegante y bien diseñada. Emplea además comandos por voz mediante *Siri AI*; Android aun podría mejorar más su usabilidad ya que se percibe como una mezcla de buenas ideas sin un fuerte diseño general; BlackBerry es usable para llamar, mensajería y email, pero pobre en cualquier otra tarea; Windows Phone apuesta por estos apartados, ofreciendo facilidad de uso y buen diseño.

- **Navegación web**: La experiencia de navegación en iOS es excelente, la característica del *Reader* reduce el desorden y el uso de datos y muestra sólo el texto del artículo. Su pega es que no soporta *Adoble Flash* (*iSwifter* es un recomendado navegador Flash en iOS); En Android la experiencia es también excelente, muy rápida y soporta *Adobe Flash* y 16 pestañas; En BlackBerry el navegador *Wikitude* ofrece una navegación fluida y soportando *Flash* y atajos de teclado. La pega es el pequeño tamaño de sus pantallas; Windows Phone por su parte es un navegador preparado para *HTML5*, pero que aun no soporta *Adobe Flash*.

- **Email & Mensajería**: iOS dispone de una buena implementación, *iMessage* permite enviar texto, imágenes y videos de forma gratuita entre *iPhones*, *iPod Touches* y *iPads*. Una posible pega es que no haya teclado físico; Android tiene integración completa con Gmail y una buena implementación que permite entrada por voz de forma precisa; BlackBerry es lo que mejor sabe hacer, dispone de teclado físico; Windows Phone también dispone de entrada por voz, de *text–to–speech* (permite que los mensajes sean leídos en voz alta) y las distintas conversaciones (Twitter, Facebook, etc.) se unen de forma fluída.

Ocio

- **Fotos & Videos**: Tanto iOS como Android destacan por ofrecer una buena experiencia en la captura de imágenes y videos mediante la cámara (con muchas *apps* para complementar), mientras que BlackBerry pierde en calidad de captura y en tamaño de pantalla y Windows Phone no dispone de video llamadas.

- **Música**: Apple apostó por *iTunes* que aunque no es de agrado para mucha gente, cumple con su cometido y permite escanear toda la música del ordenador y subirla online por solo 15 libras/año; Android en cambio le hace frente con *Google Music* que aunque inicialmente solo este disponible en EE.UU., permite almacenar para *streaming* 20,000 canciones en la nube que se sincronicen con el escritorio; A BlackBerry aun le queda mucho por mejorar, aun habiendo mejorado mucho respecto a versiones anteriores; Windows Phone viene de la mano con *Zune* que ha sido toda una gran sorpresa (bueno, rápido, fácil de usar, integración fluida, se sincroniza por WiFi, etc.).

- **Juegos**: iOS tiene una gran selección de juegos divertidos y de alta calidad con buenos gráficos; Android tiene potencial pero claramente está por detrás en juegos de alta calidad; En BlackBerry las capacidades de juego están limitadas por el sistema operativo y la librería es muy pobre; Windows Phone es uno de los factores por los que ha apostado, ya que integra *Xbox Live*, pero aun está en una etapa muy prematura.

Hardware

- **Hardware**: iOS: Excelente calidad de hardware, pero escasa variedad (elección de 1 dispositivo cada 12-18 meses); Android goza de una gran variedad de teléfonos de donde poder escoger, con gamas que van de peor a mejor calidad y ofreciendo características distintas como teclados físicos; El diseño de la BlackBerry es algo menos estético y más enfocado a ámbito empresarial (el 9990 Bold es la excepción), pero dispone de buen hardware de calidad; Windows Phone sigue el patrón de Android, pero su variedad aun es algo escasa.

- **Rendimiento**: iOS tiene buen rendimiento; Android también por lo general, especialmente con los teléfonos nuevos *dual core*; En BlackBerry existen algunos problemas de *lag* y rendimiento derivados del nuevo SO; Windows Phone buen rendimiento.

- **Batería**: En iOS (no se puede retirar la batería) y en Android existe una corta vida en la batería, a no ser que se invierta tiempo modificando las configuraciones; La BlackBerry solía durar bastante, pero con el OS7 y tamaños de pantallas mayores, impresiona algo menos; En Windows Phone la autonomía es decente y la opción de "*Battery saver*" ayuda.

- **Memoria**: En iOS la memoria no es aumentable y no se puede emplear el teléfono como medio *USB*; En Android, BlackBerry y Windows Phone, sí que es aumentable en la mayoría de los casos y en este último tampoco se puede usar como medio *USB*.

Conectividad

- **Sincronización & Backup**: Por parte de iOS, *iCloud* es gratuita y se realizan copias de seguridad y actualizaciones de los dispositivos Apple sin cables y de forma automática; En Android la sincronización es excelente con servicios Google; En BlackBerry está disponible a través de apps de terceros y *BlackBerry Protect* realiza copias automáticas de contactos, textos, favoritos y del calendario; En Windows Phone la sincronización es automática con *Zune* y existen 25GB gratuitos en la nube mediante *Microsoft Skydrive*.

- **Personalización**: iOS es una plataforma cerrada, por lo que Apple decide lo que puedes hacer (a no ser que se realice un *jailbreak*) y Android es el extremo opuesto en lo que a personalización se refiere (plataforma abierta, apps de terceras partes, ROMs personalizadas, etc.); BlackBerry es menos abierto que Android, pero más que iPhone y Windows Phone no es tan personalizable como Android, pero es flexible y muy configurable.

- **Social & Otras integraciones**: iOS integra únicamente con Twitter, mientras que Android además con Facebook, apps de Google (Maps, Voice, Mail, Calendar, Google+, etc.); En BlackBerry mediante las *Social Feeds* se actualizan de una vez Facebook, Twitter y BBM; En Windows Phone se integra Facebook, Twitter, Live Messenger y los mapas de *Bing*.

- **Actualización**: En iOS y Windows Phone la actualización se realiza de forma estandarizada sin problemas, mientras que en Android y BlackBerry la fragmentación provoca retrasos.

Otros Pros y Contras

- **iOS**:
  - **Pros:** Sincronización con Mac, un ecosistema completo de accesorios, integración y *Air Play* con dispositivos *Apple*, característica de recordatorios, *to–do list* y alertas basadas en tiempo y lugar.
  - **Contras:** Precio elevado.

- **Android**:
  - **Pros:** Cuanto más se usen los servicios de Google, más partido se le puede sacar a Android. Desbloqueo facial al mirar a la cámara, compartir contenido mediante *NFC* (usado por Android Beam), posibilidad de controlar el uso de datos en detalle.
  - **Contras:** Problemas de seguridad debido a ser una plataforma abierta.

- **BlackBerry**:
  - **Pros:** Empleo de encriptación de nivel militar con *BBM* y *BlackBerry Protect* para proteger los datos personales, muy integrado en ámbito empresarial, tecnología *NFC* para realizar acciones varias como pagos.
  - **Contras:** Pantalla pequeña, actualización problemática (dispositivos antiguos no son actualizables a la OS7), no hay soporte para *hotspot* móvil.

- **Windows Phone**:
  - **Pros:** Los *Hubs* (centros) ofrecen fácil acceso a apps y a agrupaciones suyas, *haptic feedback*, integración nativa de *Office 365*, *Word*, y *Excel*.
  - **Contras:** No se pueden enviar *sms* a números que no hayan sido etiquetados como "móviles", vinculado a *Zune*.

A continuación en la Tabla 3, un breve resumen en forma de tabla de algunas de las características a modo esquemático.

	Apple iOS 	Android 	Windows Phone 	BlackBerry OS 
Desarrollador	Apple	Google	Microsoft	Research in Motion
Smartphones populares	• Apple iPhone 4S • Apple iPhone 4	• Samsung Galaxy S II • Samsung Galaxy Note • Samsung Galaxy Nexus • HTC Sensation	• Nokia Lumia 800 • HTC Titan • LG Optimus 7	• BlackBerry Bold 9930 • BlackBerry Curve 9370
Tablets populares	• Apple iPad 2	• Samsung Galaxy Tab 10.1 • Motorola Xoom	N/A	N/A (QNX es otro sistema operativo)
Interfaz de Usuario	Basada en iconos	Iconos & widgets	Basado en "tejas" (Metro UI)	Iconos & widgets
Mercado Apps	App Store	Google Play	Windows Phone Marketplace	Blackberry App World
Apps disponibles	600,000+	500,000+	100,000+	100,000+
GPS	Vía aplicaciones	Sí, gratuito	Vía aplicaciones	Vía aplicaciones
Control parental	Sí	Vía aplicaciones	Vía aplicaciones	Vía aplicaciones
Multitarea	Pseudo	Sí	Sí	Sí
Hardware soportado	iPhone, iPod touch e iPad	Gran variedad	Variedad limitada	Variedad limitada
Soporte en la nube	iCloud	Google Sync	SkyDrive	Business Cloud Services
Explorador	Mobile Safari	Chrome	Internet Explorer 9	Bolt
Red social de juegos	Game Center	N/A	XBOX Live	N/A
Soporte Flash	No	Sí	No	No
Music Store	iTunes	Si (de Google Play)	Zune	BlackBerry Music Store
Seguridad	Si	Susceptible a malware	Si	Si

Tabla 3. Comparativa SOs móviles

A modo de resumen se observa como en calidad general compiten Android e iOS siendo el primero una plataforma abierta y el segundo un sistema cerrado, BlackBerry lleva tiempo asentado en el ámbito empresarial debido a sus medidas de seguridad y a su eficacia en términos de mail y mensajería, y por último Windows Phone que promete para vistas futuras, pero que de momento le falta algo de madurez al sistema.

3.1.4.1 Mercado

En 2006 no existían aún Android, ni iOS, ni Windows Phone, ni Bada y se vendieron 64 millones de smartphones en aquella época [30]. Actualmente se venden aproximadamente 10 veces más smartphones y los sistemas operativos con más cuota de mercado son: Android, Symbian, Apple iOS, RIM BlackBerry, MeeGo, Windows Phone y Bada [31].

La gráfica de Figura 18 muestra las ventas a nivel mundial (porcentualmente) de smartphones desde 2007 hasta 2012, según sistema operativo.

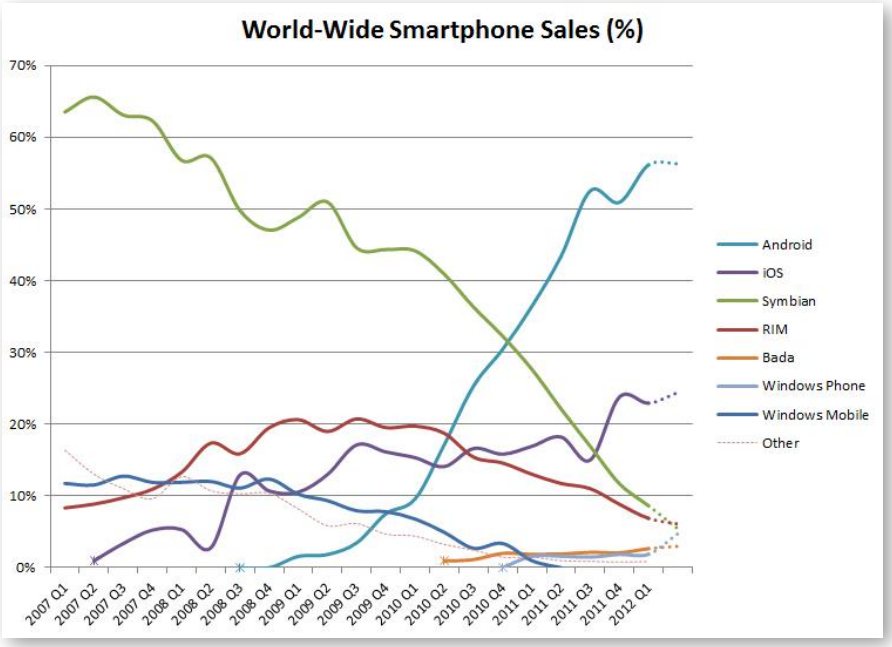


Figura 18. Ventas mundiales de Smartphones (%)

Se puede apreciar como Windows Mobile está prácticamente extinto y como Symbian y Bada siguen ese mismo camino, mientras que Android, iOS y Windows Phone (este último sobre todo) están en crecimiento.

Esta creciente popularidad de Android se puede apreciar también en los Estados Unidos, mediante un desglose de la cuota de mercado de los fabricantes de smartphones según SO (véase Figura 19). Según este informe emitido por Nielsen [32], Android tiene una cuota de mercado en EE.UU. mayor al porcentaje de cuota del resto de sistemas operativos móviles de forma conjunta. En España, concretamente, el dominio por parte de Android es incluso más notable [33] y en Europa Windows Phone tiene mejor tendencia y más cuota que la representada en este gráfico estadounidense.

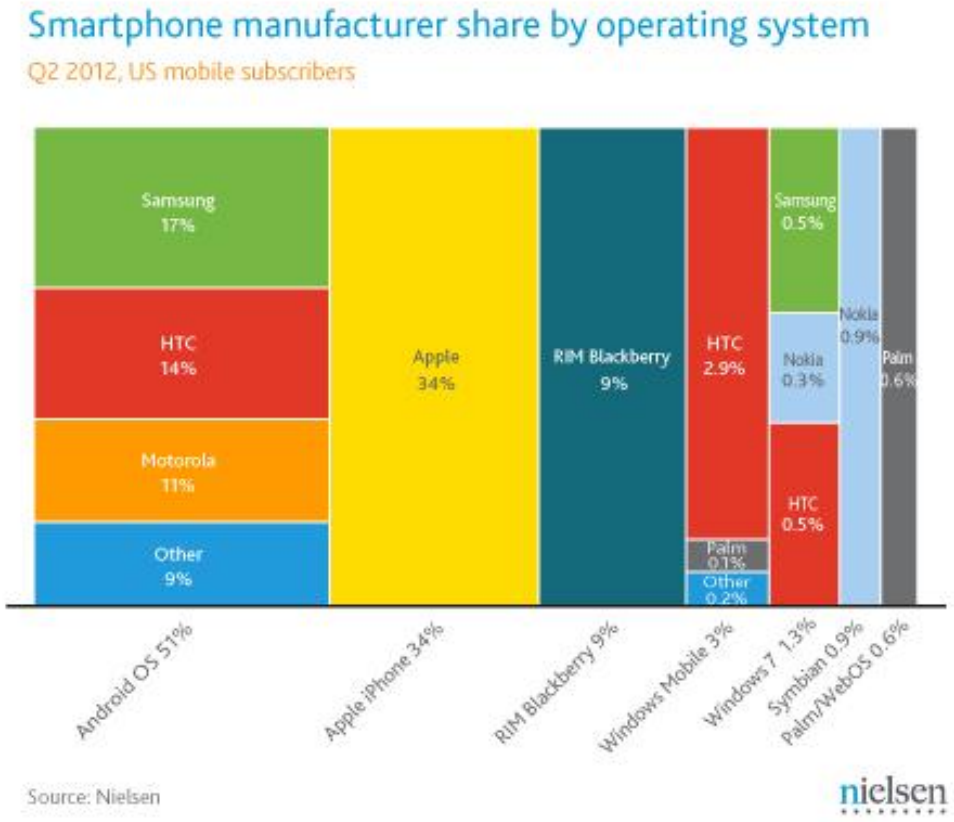


Figura 19. Q2 2012 Nielsen[6]

Por último comentar que IDC (International Data Corporation) prevé [35] un aumento de la cuota de mercado de Android del 59% en Q1 2012 a un 61% en el año entero de 2012 y una disminución al 53% hasta 2016. Que Windows Phone crezca de un 2% en Q1 2012 a un 5% en 2012 y entonces a un 19% hasta 2016. El iPhone (iOS) que descienda de 23% a 21% para el 2012, y luego vuelva a decaer a un 19% hasta 2016. BlackBerry por su parte se mantendrá en su 6% actual hasta 2016. A pesar de que estas predicciones sean cuestionables, no hay duda de que muestran una interesante visión de futuro. Dichas predicciones pueden apreciarse mejor en la Figura 20.

[6] *El tamaño de los rectángulos no representa cada uno de los porcentajes, ya que de otro modo no podría apreciarse el gráfico correctamente.* [34]

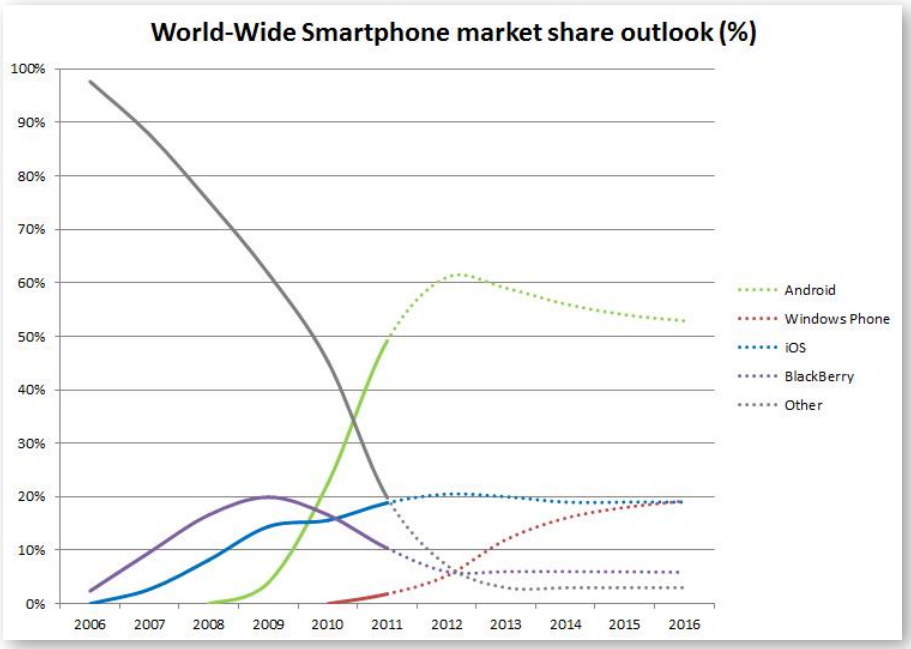


Figura 20. Previsión de cuota de mercado mundial por IDC para 2016

Teniendo en cuenta la gran aceptación a nivel mundial del sistema operativo Android, de ser una plataforma abierta que no impone restricciones a la hora de la creación de aplicaciones, de tener soporte para una gran variedad de dispositivos y de ser relativamente asequible conseguir un terminal (en comparación con iOS por ejemplo) para realizar pruebas con el software a desarrollar; se ha decidido emplear Android frente al resto en la creación de este Trabajo de Fin de Grado.

3.2 Protocolo de comunicación

La computación móvil implica un nuevo estilo de computación que es altamente dependiente del contexto. Los recursos disponibles suelen estar sujetos a fuertes restricciones y la comunicación no sólo es poco fiable, sino que también es muy impredecible al depender fuertemente de las características del entorno local.

3.2.1 Problemas de movilidad

El mecanismo habitual empleado para aplicaciones móviles que requiere de actualizaciones instantáneas, es el de sondear (*poll*) de forma periódica en busca de datos. Teniendo en cuenta las características anteriormente mencionadas, este enfoque se vuelve tremendamente ineficiente, ya que conduce al servidor a la contención de recursos, a la sobrecarga de la red y a la congestión. A esto se le añade que los dispositivos móviles dependen de una fuente finita de energía, por lo que las peticiones innecesarias de información deberían ser evitadas.

La alternativa es un método más avanzado y eficiente: el *server push* (empuje de información por parte del servidor), de esta forma el proceso de envío de información se automatiza, sin necesidad de que lo hubiera solicitado previamente el cliente. Los servicios *push* se basan normalmente en preferencias de información prestablecidas, o lo que también se conoce como paradigma *publish/subscribe* [36]. Dicho paradigma puede ofrecer un alto grado de desacoplamiento entre los elementos de una aplicación distribuida. Es por esta razón por la que *publish/subscribe* es considerada muy adecuada para entornos dinámicos como el móvil [37], donde el conjunto de componentes se somete a continuas reconfiguraciones. En años recientes, varios prototipos han sido presentados demostrando la integración del mecanismo de enrutamiento *publish/subscribe* tanto con redes de sensores [38] como con plataformas móviles [39].

3.2.1.1 Patrón *Publish/Subscribe*

Un sistema *publish/subscribe* (*pub/sub*) es un modelo de comunicación *middleware* donde las entidades implicadas se pueden dividir en dos grandes roles: editores (*publishers*) y suscriptores (*subscribers*) [40]. Los editores producen la información sin planear el envío de los mensajes de forma directa a suscriptores específicos, en su lugar, publican dicha información en uno o más canales de datos que se encuentren disponibles. De forma similar los suscriptores consumen la información, recibiendo únicamente los mensajes que son de interés de los canales a los que están suscritos. El elemento que actúa de mediador coordinando las suscripciones y controlando las colas para garantizar que la información se publica y se recibe correctamente, es normalmente un evento de entidad de intermediación.

Existen tres tipos principales de patrones *pub/sub* que pueden ser usados para crear un mecanismo para enviar los datos: "Basado en el tema" (*Topic-Based*), "Basado en la transmisión" (*Broadcast-Based*), y "Basado en el contenido" (*Content-Based*) [41]:

- *Topic-Based*: Los elementos del sistema intercambian información a través de un conjunto de listas de *topics* predefinidas o nombres de canales lógicos, normalmente conocidos de antemano. Todos los suscriptores de un *topic* recibirán todos los mensajes publicados en dicho *topic*.

- *Broadcast*-Based: Todo mensaje se transmite a todo el sistema. Cada una de las entidades que reciba la información deberá inspeccionarla y en el caso de que esté suscrita al tema del mensaje, procesarla.

- *Content*-Based: En este enfoque, las suscripciones están directamente relacionadas con contenido específico de información, ofreciendo así el modelo *pub/sub* más flexible y versátil.

Teniendo en cuenta que el servidor y el *backend* sobre el que se trabajará en este proyecto ya están instaurados, se considerará por tanto el tipo de patrón *Topic-Based* al ser el que ya está establecido. Se consideró que los sistemas *Topic-Based* eran más apropiados para redes de sensores inalámbricas (*WSNs*) debido a su simplicidad comparándolos con otros tipos de *pub/sub*. En la Figura 21 se muestra el esquema gráfico de un sistema *pub/sub* basado en la temática.

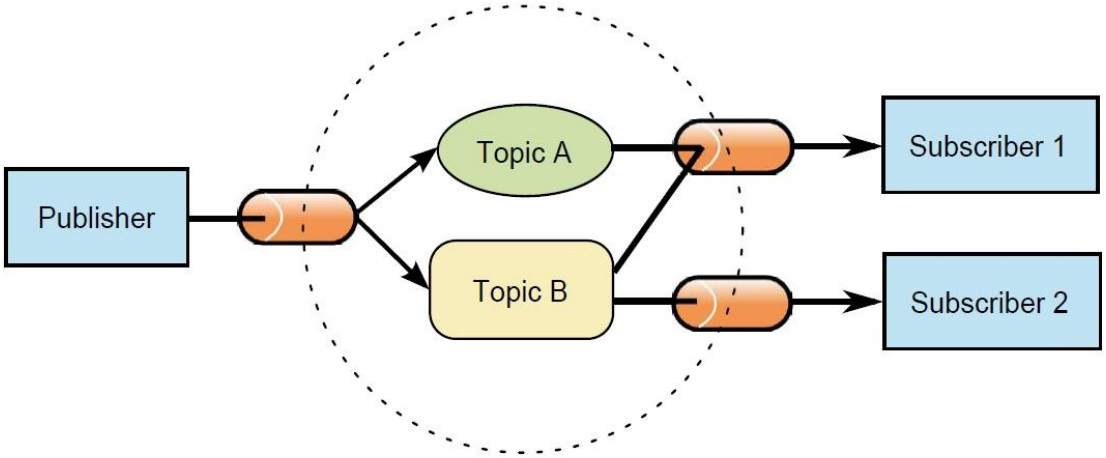


Figura 21. Patrón de diseño *publish/subscribe* basado en *topic*

La transmisión de mensaje unidireccional que caracteriza los modelos *publish/subscribe* proporciona una comunicación asíncrona entre los distintos elementos. Los editores no se bloquean mientras producen eventos, y de forma similar los suscriptores pueden recibir datos asíncronamente mientras desempeñan otras tareas concurrentes.

La inherente comunicación anónima y el dinamismo de modelos *publish/subscribe* permite el desarrollo de sistemas flexibles, muy adaptables a frecuentes conexiones y desconexiones de los nodos móviles, tal y como ocurre en una red móvil [42].

3.2.2 Descripción del Sistema

Como se comentó anteriormente, el *backend* ya venía establecido y para ello se seleccionó en su momento un entorno doméstico real para probar su efectividad. Ese dominio permitía no solo evaluar el rendimiento general del sistema completo, sino también juzgar hasta que grado los usuarios estaban dispuestos a aceptar la red de sensores. El cometido era monitorizar los movimientos y actividades de ancianos residiendo de forma independiente. Varios investigadores emplearon previamente redes de sensores para estudiar patrones de comportamiento de vivir independientemente [43].

En esta sección se presenta la tecnología *pub/sub* seleccionada como plataforma mediadora del sistema.

3.2.2.1 MQTT ("*Message Queuing Telemetry Transport*")

MQTT es sinónimo de "MQ *Telemetry Transport*". Es un protocolo abierto de mensajes *publish/suscribe* [44] extremadamente simple y ligero, diseñado para dispositivos limitados y de bajo ancho de banda, alta latencia o redes poco fiables. Los principios de diseño son intentar minimizar el ancho de banda de la red y los recursos requeridos por el dispositivo, mientras garantiza la fiabilidad y cierto grado de garantía en la entrega. Estos principios también surgen como base para el protocolo ideal del mundo de dispositivos conectados que acontece, "*machine–to–machine*" (M2M) o "el Internet de las cosas". Por supuesto, también para aplicaciones móviles donde el ancho de banda y la batería están muy solicitados. Una ventaja importante de MQTT es que la complejidad del sistema reside en la implementación del *broker*, permitiendo clientes simples y ligeros.

Un aspecto relevante de sistemas de información distribuidos es la fiabilidad. Para asegurar una entrega oportuna de los mensajes, MQTT soporta básica calidad de servicio (*QoS*) "extremo a extremo" [45]. Se definen tres niveles de *QoS* dependiendo de la fiabilidad requerida por la conexión de datos, pero un nivel de *QoS* más alto implica requerimientos mayores de latencia y ancho de banda.

El *broker* también supervisa el estado de la conexión del cliente mediante un mecanismo que establece el máximo intervalo de tiempo que puede transcurrir entre dos mensajes recibidos por el cliente.

3.2.2.2 Arquitectura

En este capítulo se ofrece una descripción más detallada del sistema ya implantado. El diseño básico del *framework* está dividido en tres áreas principales: la red de sensores, el servidor de administración centralizada y el entorno móvil de monitorización.

La red de sensores que se empleó fue elegida debido a dos criterios principales: facilidad de instalación y mínima intrusión. Los nodos inalámbricos fueron fabricados por *RFM* y su *firmware* estándar fue diseñado para incluir un protocolo de red que permitiese una larga vida para la batería. La comunicación inalámbrica entre los nodos puede alcanzar transmisiones de datos de 4.8 kb/s. La red está configurada en una topología en estrella (véase Figura 22), donde el punto de acceso es una entrada central (*gateway*) adjunta al servidor y el campo de todos los nodos se encuentra en su radio. Debido a la buena fiabilidad y precisión que esta red ofrece, otros sistemas similares han sido desarrollados previamente en base a esta tecnología [47].

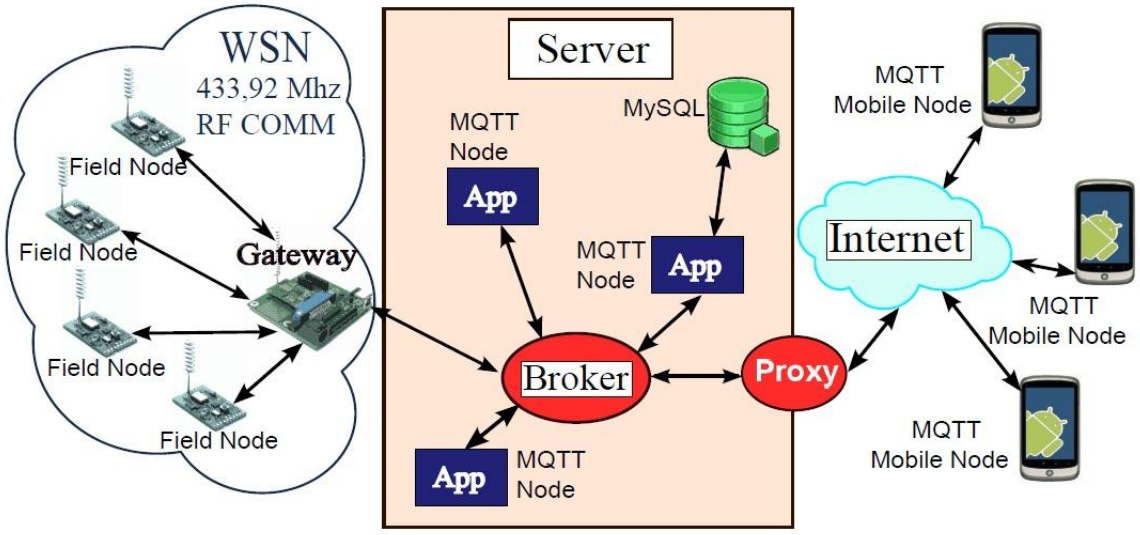


Figura 22. Arquitectura propuesta (WSN, Server y entorno móvil)

En servidor de administración centralizada puede entenderse como el puente necesario entre la red de sensores y los terminales móviles. Diferentes funcionalidades clave del sistema son tratadas por este módulo, teniendo cada componente un papel bien definido. Uno de los componentes es usado como interfaz entre el servidor y la red de sensores. El componente *broker* proporciona el servicio *publish/subscribe* a la aplicación. También existe un componente encargado del almacenamiento de datos usando una base de datos *MySQL*. Tal y como se ha mencionado, la comunicación entre estos distintos componentes está siempre basada en el modelo *publish/subscribe*.

Tal y como se vio en el apartado 3.1 se ha elegido el sistema operativo *open-source* Android para la monitorización móvil. Dicha elección fue debida entre otras razones a la facilidad de integración con el resto del sistema (mismo lenguaje base), a su inherente naturaleza multitarea, a su desarrollo mediante el *SDK* gratuito, *open-source* y soportado oficialmente para varias plataformas, y a la posibilidad de emplear notificaciones *pop-up*.

3.3 Aplicaciones similares existentes

Actualmente existe en el mercado tecnológico una diversa variedad de herramientas, tanto sobre los distintos sistemas operativos (Android, iOS, etc.), como sobre hardware independiente, que permiten telemonitorizar o controlar a distancia ciertos elementos. A continuación se expondrán, por tanto, algunos ejemplos de las herramientas que existen para monitorizar entornos.

3.3.1 Aplicaciones Android

En esta sección se tratarán algunas de las aplicaciones programadas en base al sistema operativo Android que persigan la temática de telemonitorización en cualquiera de sus vertientes.

- **Media Remote** [48]**:** Es una aplicación gratuita que funciona con determinados dispositivos de la casa Sony, sobre todo en ámbito de entretenimiento, tales como reproductores *Blue-ray Disc (TM)*, dispositivos *Bravia*, etc. La aplicación permite accionar el dispositivo de entretenimiento a través del dispositivo móvil, o simplemente mostrar la información relacionada con el contenido que se está viendo en ese momento. Requiere que ambos dispositivos estén conectados a la misma red inalámbrica.

Figura 23. Media Remote - Captura Pantalla

- **Philips MyRemote** [49]**:** Es una aplicación gratuita inteligente que ayuda a organizar la experiencia de visualización como se guste. Puede sustituir el mando convencional y ofrece opciones adicionales como introducción sencilla de texto o compartición de archivos multimedia, entre otras. Muy similar a la anterior, sólo que en este caso en lugar de Sony está dedicada a productos Philips (*Net TV*, *Smart TV*).

Figura 24. Philips MyRemote - Captura Pantalla

- **AZRemote** [50]**:** Esta aplicación de pago (2,36€) permite controlar todas las funcionalidades de un receptor satélite *AZBox* desde el terminal Android. Es compatible con los decodificadores *AZBox HD Elite*, *Premium* y *Premium +*. Para poder usar la aplicación es necesario configurar la dirección IP del receptor correspondiente para poder controlarlo, por lo que se podrá usar a modo de mando a distancia desde cualquier punto con acceso a *WiFi* o tarifa de datos.

Figura 25. AZRemote – Captura Pantalla

- **EagleEyes** [51]**:** Esta aplicación de pago (4,06€ o gratuita en su versión *Lite*) cumple con la función básica de vigilancia remota a través del control táctil de la pantalla del teléfono. Entre los dispositivos compatibles se incluyen *AVTECH DVR* y las cámaras *IP AVTECH*. Los usuarios podrán comprobar visualización en vivo y cambio de canal. Otra característica remarcable es el *Push* de video que permite avisar al cliente de cambios obtenidos por la cámara, sin necesidad de que este último realice peticiones innecesarias para comprobar el estado**.**

Figura 26. EagleEyes – Captura Pantalla

- **mydlink+** [52]**:** Esta aplicación de pago (0,81€ o gratuita en su versión *Lite*) permite al igual que *EagleEyes* ver en vivo el video de la cámara de vigilancia *mydlink* bien sea empleando *WiFi* o *3G*. Está ideada sobre todo para su visualización en *Tablets*, aun funcionando también para *smartphones*. Posibilidad de realizar *snapshots*. Incluye también la tecnología *Push*, de modo que es posible configurar las cámaras para que detecten movimientos y produciéndose una intrusión, se envié una alerta por correo electrónico.

Figura 27. mydlink+ – Captura Pantalla

3.4 Herramientas

A continuación se procederá a introducir las herramientas empleadas en este Proyecto de Fin de Grado, resumiendo sus características principales.

3.4.1 Android SDK

El Android SDK (Software Development Kit), descargable desde la página oficial de Android, comprende una serie de herramientas de desarrollo necesarias para crear aplicaciones para este sistema operativo. Dicho SDK está compuesto por paquetes modulares que se pueden descargar de forma separada usando el *Android SDK Manager*. Algunos de los paquetes disponibles más remarcables son los siguientes [53]:

- SDK Tools: Contiene herramientas de depuración y de pruebas, junto con otras utilidades requeridas para desarrollar una aplicación. Conviene mantenerlo actualizado (al instalar el *SDK starter package* se obtiene la última versión).

- SDK Platform–tools: Contiene herramientas de desarrollo y depuración dependientes de la plataforma, que soportan las últimas características de la plataforma Android y que son típicamente actualizadas cuando una nueva plataforma entra a estar disponible. Estas herramientas son compatibles con versiones de plataformas más antiguas.

- Documentación: Una copia offline de la última documentación de las APIs de la plataforma Android.

- Ejemplos para el SDK: Una colección de apps de ejemplo basándose en las APIs de la plataforma. Ejemplos pensados sobre todo para desarrolladores iniciados en código de Android.

- Google Play Billing: Proporciona las librerías estáticas y los ejemplos que permiten integrar los servicios de facturación en la app.

- Google Play Licensing: Proporciona las librerías estáticas y los ejemplos para llevar a cabo la verificación de la licencia de la app a la hora de distribuirla.

Hay algunos casos en los que un paquete del SDK pueda requerir una versión específica mínima de otro paquete o del *SDK Tools*. Por ejemplo podría haber una dependencia entre el *Plugin ADT* para Eclipse y el paquete *SDK Tools* (por ejemplo, *ADT 8.x* requiere *SDK Tools r8*).

Comentar además que para que la tarea de desarrollo sea lo más sencilla posible, se recomienda instalar el driver USB (*Windows*) [54] y el *plug-in* para *Eclipse*, incluido en el kit de desarrollo, el cual integra las características del SDK dentro del entorno del mismo *IDE*.

3.4.2 Eclipse

"*Una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular.*"

Proyecto Eclipse

Eclipse es un entorno integrado de desarrollo (IDE) multiplataforma para crear aplicaciones clientes de cualquier tipo. Es libre y fue creado originalmente por IBM, aunque ahora lo desarrolla la Fundación Eclipse, una organización independiente sin ánimo de lucro que apoya una comunidad de código abierto. La aplicación más importante que ha sido realizada con este entorno es el afamado entorno de desarrollo integrado Java, llamado Java Development Toolkit (JDT). Este es el entorno de desarrollo que se está convirtiendo en el estándar de facto para Java, de hecho otros IDE's comerciales como JBuilder han anunciado que su próxima versión se basará en Eclipse.

Eclipse no es tan sólo un IDE, se trata de un framework ampliable mediante módulos (en inglés plug-in). Estos módulos o plugins proporcionan más funcionalidades, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Incluso hay plugins que permiten que el entorno de desarrollo soporte otros lenguajes además de Java, como por ejemplo PHP, Perl, etc.

Los componentes gráficos de Eclipse están basados en un juego de herramientas de tercera generación para Java de IBM llamado SWT, que mejora los de primera y segunda generación de Sun. La interfaz de usuario de Eclipse cuenta con una capa intermedia de interfaz gráfica (GUI) llamada JFace, lo que simplifica la creación de aplicaciones basadas en SWT.

4 Objetivos

"En realidad no trato de destruir a Microsoft: eso será sólo un efecto colateral no intencionado".

Linus Torvalds – "padre" de Linux

Es en este capítulo en el que se presenta la meta y los objetivos principales a perseguir en este Trabajo de Fin de Grado, enfocado en el ámbito de la telemonitorización mediante Android.

El objetivo general de este Trabajo de Fin de Grado es:

- Desarrollar una aplicación que actúe de *frontend* para la telemonitorización de una red de sensores ya implantada y encargada de la supervisión de personas ancianas residiendo en un domicilio.

Además de dicho objetivo principal, existen una serie de objetivos específicos a abordar:

- Estudio de aplicaciones dirigidas a *smartphone* con relación en el ámbito de la telemonitorización/vigilancia.

- Proceso de aprendizaje para el desarrollo de aplicaciones sobre la plataforma Android (SDK), de forma que sean válidas también para futuras versiones de la misma.

- Análisis de la red de sensores y del *backend* ya existente para desarrollar la aplicación Android en consecuencia.

- Estudio general de *frameworks*/mecanismos para la comunicación *push* entre terminales, y de MQTT en particular.

- Diseño de un protocolo de comunicación para el envío de mensajes y eventos entre el *frontend* y el *backend*.

- Diseño de un método de evaluación válido para comprobar que los datos llegan en "*soft realtime*" [55].

- Verificación de que dicha aplicación desarrollada funcione correctamente y su posterior despliegue al ámbito real, es decir, a un *smartphone*.

5 Desarrollo de la aplicación

"Hay dos maneras de diseñar software: una es hacerlo tan simple que sea obvia su falta de deficiencias, y la otra es hacerlo tan complejo que no haya deficiencias obvias".

C. A. R. Hoare

A continuación se especifica todo lo relacionado con el desarrollo de la aplicación, comenzando con el detalle de los requisitos establecidos por el usuario (con los consecuentes requisitos software), seguido de los casos de uso y de la arquitectura, concluyendo con la implementación final del *frontend*.

Tal y como se vio en el apartado 2.3 "Ciclo de vida del software", el proyecto se ha abordado desde un modelo en cascada, y es de esta forma como a su vez se ha estructurado la memoria. Es decir, dentro de esta sección también se persigue una lógica descendente en la que cada uno de los apartados es continuación del anterior. No podrían especificarse, por ejemplo, los requisitos software si previamente no se hubiesen detallado los requisitos de usuario.

5.1 Análisis

Esta fase se considera sumamente importante en el proceso de desarrollo de un sistema interactivo, al servir de base para comprender y concretar el sistema, y conocer de esta forma los elementos que conforman el contexto del problema. Los productos del análisis a desarrollar serán los siguientes: Requisitos de la aplicación (RUs y RSs) y casos de uso.

5.1.1 Requisitos de la aplicación

Para poder proseguir con el desarrollo de la aplicación ha de existir una representación documentada de las condiciones o capacidades que ha de cumplir la misma. Dichos requisitos se recogerán en distintas tablas, conteniendo toda la información necesaria para su correcta comprensión. Véase apartado 2.4 para el formato de la tabla y nomenclatura de los requisitos.

5.1.1.1 Requisitos de usuario

En este grupo de requisitos se encuentran los definidos por el usuario, es decir, aquellas características demandadas por el cliente que tienen como objetivo plantear el problema y delimitar qué es lo que ha de cumplir la aplicación. Dichos requisitos son:

ID: RU-C01			
Nombre:	Almacenar activaciones		
Tipo requisito:	Capacidad		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	La información sobre las activaciones de cada sensor deberá almacenarse en el sistema.		

Tabla 4. RU–C01 / Almacenar activaciones

ID: RU-C02					
Nombre:	Mostrar sensores				
Tipo requisito:	Capacidad				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	El sistema debe permitir listar los sensores que han sido activados.				

Tabla 5. RU–C02 / Mostrar sensores

ID: RU-C03					
Nombre:	*Splash screen*				
Tipo requisito:	Capacidad				
Necesidad:	[ ]Esencial	[X]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[ ]Media	[X]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	El sistema debe mostrar una pantalla de bienvenida o *splash screen* nada más iniciar la aplicación.				

Tabla 6. RU–C03 / *Splash screen*

ID: RU-C04					
Nombre:	Información de conexión				
Tipo requisito:	Capacidad				
Necesidad:	[ ]Esencial	[X]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[X]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	El sistema debe permitir mostrar información sobre la conexión actual.				

Tabla 7. RU–C04 / Información de conexión

ID: RU-C05					
Nombre:	Resaltar última activación				
Tipo requisito:	Capacidad				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[X]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	El sistema debe resaltar entre los sensores, el que haya sido el último en activarse.				

Tabla 8. RU–C05 / Resaltar última activación

ID: RU-C06					
Nombre:	Filtrar por entorno				
Tipo requisito:	Capacidad				
Necesidad:	[ ]Esencial	[X]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[ ]Media	[X]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	El sistema debe permitir generar la lista de sensores según el entorno al que pertenezcan.				

Tabla 9. RU–C06 / Filtrar por entorno

ID: RU-C07					
Nombre:	Detener servicio				
Tipo requisito:	Capacidad				
Necesidad:	[ ]Esencial	[X]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[ ]Media	[X]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	El sistema debe permitir la parada de la aplicación con el fin no recibir más eventos de sensores.				

Tabla 10. RU–C07 / Detener servicio

ID: RU-C08					
Nombre:	Notificaciones				
Tipo requisito:	Capacidad				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[X]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	El sistema debe permitir notificar al usuario la llegada de una nueva activación.				

Tabla 11. RU–C08 / Notificaciones

ID: RU-C09					
Nombre:	Gráficas				
Tipo requisito:	Capacidad				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[X]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	El sistema debe permitir mostrar una gráfica para cada sensor donde se muestre si ha habido activaciones en un intervalo de tiempo dado.				

Tabla 12. RU–C09 / Gráficas

ID: RU-C10					
Nombre:	Parámetros gráficas				
Tipo requisito:	Capacidad				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[X]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	La resolución de las gráficas (número de puntos), así como el intervalo de tiempo deben de poder ser configurables.				

Tabla 13. RU–C10 / Parámetros gráficas

ID: RU-C11					
Nombre:	Dirección del servidor				
Tipo requisito:	Capacidad				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[X]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	La dirección del servidor a la que conectarse debe de poder ser configurable.				

Tabla 14. RU–C11 / Dirección del servidor

ID: RU-C12					
Nombre:	Tema de suscripción				
Tipo requisito:	Capacidad				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[X]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	El tema o *topic* al que suscribirse para recibir eventos debe poder ser configurable.				

Tabla 15. RU–C12 / Tema de suscripción

ID: RU-C13					
Nombre:	Entorno				
Tipo requisito:	Capacidad				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[X]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	La elección del entorno debe poder ser configurable.				

Tabla 16. RU–C13 / Entorno

ID: RU-C14					
Nombre:	Extensibilidad de idiomas				
Tipo requisito:	Capacidad				
Necesidad:	[ ]Esencial	[X]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[ ]Media	[X]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	Además del idioma base, el sistema deberá poder extenderse a cualquier otro idioma en un futuro.				

Tabla 17. RU–C14 / Extensibilidad de idiomas

ID: RU-R01					
Nombre:	Idioma de la aplicación				
Tipo requisito:	Restricción				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	El sistema deberá estar en inglés.				

Tabla 18. RU–R01 / Idioma de la aplicación

ID: RU-R02					
Nombre:	Compatibilidad Android				
Tipo requisito:	Restricción				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	El sistema deberá poder funcionar con normalidad en cualquier versión de Android superior a la 2.1 (ésta inclusive).				

Tabla 19. RU–R02 / Compatibilidad Android

ID: RU-R03					
Nombre:	*Portrait / Landscape*				
Tipo requisito:	Restricción				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario
Descripción:	El sistema deberá ejecutarse en *portrait* (en vertical) en todas las vistas, salvo en las gráficas que se permite también *landscape* (apaisado).				

Tabla 20. RU–R03 / *Portrait / Landscape*

5.1.1.2 Requisitos de software

En este otro grupo de requisitos se encuentran los que en base a los requisitos de usuario, definen como se comporta el sistema de forma más consistente y completa. Dichos requisitos son los siguientes:

ID: RS-F01					
Nombre:	Información de activación				
Tipo requisito:	Funcional				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Analista
Descripción:	Cada activación almacenada tendrá un identificador único (asociado al sensor), el valor de la activación y la fecha en la que se activó.				

Tabla 21. RS–F01 / Información de activación

ID: RS-F02					
Nombre:	Lista de sensores				
Tipo requisito:	Funcional				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	Los sensores se mostrarán en la lista en orden de llegada, en el caso de haber recibido al menos una activación.				

Tabla 22. RS–F02 / Lista de sensores

ID: RS-F03					
Nombre:	Valores de la lista				
Tipo requisito:	Funcional				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	La información de cada sensor de la lista estará compuesta de un identificador hardware único, del tipo de sensor, del lugar y de la fecha de última activación.				

Tabla 23. RS–F03 / Valores de la lista

ID: RS-F04					
Nombre:	Mensaje defecto de lista				
Tipo requisito:	Funcional				
Necesidad:	[ ]Esencial	[ ]Deseable	[X]Opcional		
Prioridad:	[ ]Alta	[ ]Media	[X]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	El sistema deberá mostrar un mensaje por defecto en el caso de no haberse activado ningún sensor.				

Tabla 24. RS–F04 / Mensaje defecto de lista

ID: RS-F05					
Nombre:	Acción *Splash screen*				
Tipo requisito:	Funcional				
Necesidad:	[ ]Esencial	[X]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[ ]Media	[X]Baja	**Estabilidad:**	Alta
Verificabilidad:	[ ]Alta	[X]Media	[ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	El *splash screen* deberá durar 5 segundos (o nada al pulsar en la pantalla), dando paso a la lista de sensores.				

Tabla 25. RS–F05 / Acción *Splash screen*

ID: RS-F06					
Nombre:	Luz de conexión				
Tipo requisito:	Funcional				
Necesidad:	[ ]Esencial	[X]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[ ]Media	[X]Baja	**Estabilidad:**	Alta
Verificabilidad:	[ ]Alta	[X]Media	[ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	El sistema mostrará una imagen indicadora de la conexión, la cual estará a verde si hubiese una conexión establecida o roja en caso contrario.				

Tabla 26. RS–F06 / Luz de conexión

ID: RS-F07					
Nombre:	Diálogo de conexión				
Tipo requisito:	Funcional				
Necesidad:	[ ]Esencial	[X]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[ ]Media	[X]Baja	**Estabilidad:**	Alta
Verificabilidad:	[ ]Alta	[X]Media	[ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	El sistema debe permitir mostrar un cuadro de diálogo con la información básica de la conexión: *ip* del servidor, puerto, *topic* suscrito.				

Tabla 27. RS–F07 / Diálogo de conexión

ID: RS-F08					
Nombre:	Fondo de la activación				
Tipo requisito:	Funcional				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[ ]Alta	[X]Media	[ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	El sistema debe representar el último sensor activo con un color distinto de fondo, respecto al resto de sensores de la lista.				

Tabla 28. RS–F08 / Fondo de la activación

ID: RS-F09					
Nombre:	Notificaciones entorno				
Tipo requisito:	Funcional				
Necesidad:	[ ]Esencial	[X]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[ ]Media	[X]Baja	**Estabilidad:**	Alta
Verificabilidad:	[ ]Alta	[X]Media	[ ]Baja	**Fuente:**	Analista
Descripción:	Una vez seleccionado un entorno, sólo se mostrarán notificaciones de los sensores que compartan ese mismo entorno.				

Tabla 29. RS–F09 / Notificaciones entorno

ID: RS-F10					
Nombre:	Cerrar conexiones abiertas				
Tipo requisito:	Funcional				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[X]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[ ]Alta	[X]Media	[ ]Baja	**Fuente:**	Analista
Descripción:	Al pulsar el botón de salir/detener, el sistema deberá de cerrar la interfaz y las conexiones pendientes, con el servidor y con la base de datos para liberar recursos.				

Tabla 30. RS–F10 / Cerrar conexiones abiertas

ID: RS-F11					
Nombre:	Notificaciones barra de estado				
Tipo requisito:	Funcional				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Analista
Descripción:	Estando activada la opción de notificaciones, se mostrará una notificación en la barra de estado del dispositivo, al recibir una activación si la aplicación se encuentra en segundo plano.				

Tabla 31. RS–F11 / Notificaciones barra de estado

ID: RS-F12					
Nombre:	Notificaciones sonido/vibración				
Tipo requisito:	Funcional				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Analista
Descripción:	Estando o no activada la opción de notificaciones, se producirá un sonido en el dispositivo al recibir una activación si la aplicación se encuentra en primer plano.				

Tabla 32. RS–F12 / Notificaciones sonido/vibración

ID: RS-F13					
Nombre:	Detalles gráficas				
Tipo requisito:	Funcional				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	Cada punto de la gráfica equivale a un intervalo de tiempo, que tomará valor 1 si ha existido una activación de ese sensor en ese tiempo o 0 en caso contrario.				

Tabla 33. RS–F13 / Detalles gráficas

ID: RS-F14					
Nombre:	Preferencias				
Tipo requisito:	Funcional				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	El sistema deberá ofrecer una vista de preferencias cuyos cambios se almacenarán de forma persistente en el sistema con los siguientes campos: activación de notificaciones, intervalo de tiempo (gráfica), número de puntos (gráfica), servidor, *topic* y entorno.				

Tabla 34. RS–F14 / Preferencias

ID: RS-F15					
Nombre:	*Publish arrived*				
Tipo requisito:	Funcional				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Analista
Descripción:	El sistema deberá ser capaz de transformar la información recibida por el sensor a través del canal en un formato legible por el sistema y el usuario.				

Tabla 35. RS–F15 / *Publish arrived*

ID: RS-NF01					
Nombre:	Memoria de la aplicación				
Tipo requisito:	Consumo de recursos				
Necesidad:	[ ]Esencial	[X]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[X]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[ ]Alta	[X]Media	[ ]Baja	**Fuente:**	Analista
Descripción:	La aplicación deberá de ocupar en memoria menos de 2MB con el fin de evitar una muerte prematura por parte del SO.				

Tabla 36. RS–NF01 / Memoria de la aplicación

ID: RS-NF02					
Nombre:	Versión SDK Android				
Tipo requisito:	Restricciones				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Analista
Descripción:	El sistema deberá de implementarse con el SDK 7 como versión mínima, para garantizar compatibilidad entre versiones.				

Tabla 37. RS–NF02 / Versión SDK Android

ID: RS-NF03					
Nombre:	Base de Datos ligera				
Tipo requisito:	Restricciones				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Analista
Descripción:	El sistema empleará una base de datos SQLite (< 200KB vacía) como capa de persistencia de datos.				

Tabla 38. RS–NF03 / Base de Datos ligera

ID: RS-NF04					
Nombre:	Bloqueo rotación				
Tipo requisito:	Requisitos de interfaz				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[X]Alta	[ ]Media	[ ]Baja	**Fuente:**	Analista
Descripción:	El sistema bloqueará la rotación a modo apaisado a las siguientes actividades: *SplashScreen*, *MainActivity* y *PreferencesActivity*.				

Tabla 39. RS–NF04 / Bloqueo rotación

ID: RS-NF05					
Nombre:	Separación cadenas de texto				
Tipo requisito:	Restricciones				
Necesidad:	[ ]Esencial	[X]Deseable	[ ]Opcional		
Prioridad:	[ ]Alta	[ ]Media	[X]Baja	**Estabilidad:**	Alta
Verificabilidad:	[ ]Alta	[X]Media	[ ]Baja	**Fuente:**	Analista
Descripción:	Las cadenas de texto deberán accederse desde un fichero *xml* externo con el fin de poder implantar con facilidad en un futuro la adición de variedad de lenguajes.				

Tabla 40. RS–NF05 / Separación cadenas de texto

ID: RS-NF06					
Nombre:	XML Idioma				
Tipo requisito:	Restricciones				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[ ]Alta	[X]Media	[ ]Baja	**Fuente:**	Analista
Descripción:	El sistema deberá de contener un archivo *xml* con las cadenas de texto en inglés.				

Tabla 41. RS–NF06 / XML Idioma

ID: RS-NF07					
Nombre:	Tiempo entre recepción y UI				
Tipo requisito:	Rendimiento				
Necesidad:	[X]Esencial	[ ]Deseable	[ ]Opcional		
Prioridad:	[X]Alta	[ ]Media	[ ]Baja	**Estabilidad:**	Alta
Verificabilidad:	[ ]Alta	[ ]Media	[X]Baja	**Fuente:**	Analista
Descripción:	El tiempo entre la recepción de la activación de un sensor y su actualización en la Interfaz del Usuario deberá ser inferior a 1 segundo.				

Tabla 42. RS–NF07 / Tiempo entre recepción y UI

5.1.1.3 Trazabilidad: RS <– RU

Tras la especificación de los requisitos y tal y como se comentó en el apartado anterior, es necesario comprobar que existe una correspondencia entre lo demandado por el usuario y lo que realmente se ha implementado en la herramienta.

Para ello se hará uso de la matriz de trazabilidad indicada en la Tabla 43, en la cual todo RU debe ser desarrollado por al menos un RS, aunque es posible que haya algún RS cuya fuente no sea un RU. Tal y como se puede apreciar, se comprueba esa norma por lo que toda la funcionalidad exigida por el cliente queda respaldada.

	RU–C01	RU–C02	RU–C03	RU–C04	RU–C05	RU–C06	RU–C07	RU–C08	RU–C09	RU–C10	RU–C11	RU–C12	RU–C13	RU–C14	RU–R01	RU–R02	RU–R03
RS–F01	X																
RS–F02		X															
RS–F03		X															
RS–F04		X															
RS–F05			X														
RS–F06				X													
RS–F07				X													
RS–F08					X												
RS–F09						X											
RS–F10							X										
RS–F11								X									
RS–F12								X									
RS–F13									X								
RS–F14										X	X	X	X				
RS–F15	X																
RS–NF01																	
RS–NF02																X	
RS–NF03	X																
RS–NF04																	X
RS–NF05														X			
RS–NF06															X		
RS–NF07																	

Tabla 43. Matriz de trazabilidad RU / RS

5.1.2 Casos de uso

Los casos de uso son otra de las herramientas destinadas al análisis de un proyecto software. Más concretamente, los casos de uso son una técnica de escenarios incorporada en *UML* que describe la interacción entre actores y sistema. En el caso de este TFG y debido a la escasa complejidad del mismo, el actor será único: cumplirá el cometido de dicha persona que desee monitorizar la red de sensores.

Cada uno de los casos de uso, especificará una serie de precondiciones que se han de cumplir, una serie de postcondiciones producidas tras la ejecución del mismo, un escenario principal y un escenario alternativo. El escenario principal trata la secuencia común de interacciones, mientras que el alternativo describe la ejecución en caso de error o en caso de caminos de decisión distintos al habitual.

El conjunto de casos de uso que se especifica próximamente, describe todas las posibles interacciones con el sistema. Las necesidades que van a ser plasmadas en los CUs son las siguientes:

- Obtener la información listada de los sensores activados hasta el momento
- Obtener estado de la conexión de red
- Mostrar gráfica de activaciones en función del sensor elegido
- Modificar parámetros principales del sistema
- Salir de la aplicación

Normalmente cada necesidad puede tener asociada varios CUs, en el caso de este proyecto, cada una de las necesidades se traducirá en un sólo caso de uso. A continuación se muestran los casos de uso que recogen las necesidades comentadas anteriormente:

ID: CU-01			
Título:	Obtener lista de sensores	**Actor:**	Usuario
Descripción:	La interfaz de usuario muestra la lista de sensores con los que han sido activados hasta el momento.		
Precondiciones:	La base de datos ha de contener registros de las activaciones correspondientes a cada sensor, para ello se han debido de recibir en algún momento desde que se inició la aplicación por primera vez.		
Postcondiciones:	Ninguna.		
Escenario Principal:	1. El usuario accede a la aplicación. 2. Hace clic en el *Splash Screen* o espera 5 segundos. 3. Se muestra la lista de sensores resaltando el último en haber sido activado.		
Escenario Alternativo:	1a. El usuario puede ya encontrarse en la propia aplicación. En cuyo caso, deberá pulsar la tecla "*Back*", al encontrarse en las Preferencias o en alguna de las gráficas. 2a. En el caso de haberse iniciado la aplicación, el *splash screen* no se muestra. 3a. Si no existen sensores activados (o bajo el entorno indicado) se muestra un mensaje indicándolo.		

Tabla 44. CU-01 / Obtener lista de sensores

ID: CU–02			
Título:	Obtener estado de la conexión	**Actor:**	Usuario
Descripción:	La interfaz de usuario muestra el estado de la conexión para que el usuario tenga la noción en todo momento de si se pierde o se recupera la conexión.		
Precondiciones:	Ninguna.		
Postcondiciones:	Ninguna.		
Escenario Principal:	1. El usuario accede a la aplicación. 2. Hace clic en el *Splash Screen* o espera 5 segundos. 3. La interfaz muestra si hay conexión con un indicador en verde, o en rojo en el caso de que no la hubiese.		
Escenario Alternativo:	1a. El usuario puede ya encontrarse en la propia aplicación. En cuyo caso, deberá pulsar la tecla "*Back*", al encontrarse en las Preferencias o en alguna de las gráficas. 2a. En el caso de haberse iniciado la aplicación, el *splash screen* no se muestra. 3a. El usuario puede pulsar el botón de información (?) sobre dicho indicador para obtener más información. 4a. Se muestra un diálogo emergente con información básica de la conexión: IP a la que se intenta conectar, puerto, *topic* al que se está suscrito, y si hay o no conexión.		

Tabla 45. CU–02 / Obtener estado de la conexión

ID: CU–03			
Título:	Mostrar gráfica de sensor	**Actor:**	Usuario
Descripción:	La interfaz de usuario muestra una gráfica de activaciones del sensor elegido.		
Precondiciones:	El sistema tiene almacenadas activaciones de al menos un sensor.		
Postcondiciones:	Ninguna.		
Escenario Principal:	1. El usuario accede a la aplicación. 2. Hace clic en el *Splash Screen* o espera 5 segundos. 3. Selecciona el sensor que desee. 4. El sistema muestra una gráfica en función del número de puntos y del intervalo de tiempo almacenado en las preferencias. 5. El usuario puede rotar la pantalla, para visualizar dicha gráfica en apaisado.		
Escenario Alternativo:	1a. El usuario puede ya encontrarse en la propia aplicación. En cuyo caso, deberá pulsar la tecla "*Back*", al encontrarse en las Preferencias. 2a. En el caso de haberse iniciado la aplicación, el *splash screen* no se muestra.		

Tabla 46. CU–03 / Mostrar gráfica de sensor

ID: CU–04			
Título:	Modificar parámetros del sistema	**Actor:**	Usuario
Descripción:	El usuario modifica alguno de los parámetros del sistema.		
Precondiciones:	Ninguna.		
Postcondiciones:	El sistema se actualiza según los parámetros que se hayan modificado o se mantiene invariable en caso de no haberse modificado ninguno.		
Escenario Principal:	1. El usuario accede a la aplicación. 2. Hace clic en el *Splash Screen* o espera 5 segundos. 3. Pulsa el botón de "Opciones" del dispositivo móvil. 4. El usuario elige la opción de "*Preferences*". 5. El sistema muestra cada uno de los parámetros editables. 6. El usuario modifica los parámetros que desee.		
Escenario Alternativo:	1a. El usuario puede ya encontrarse en la propia aplicación. En cuyo caso, deberá pulsar la tecla "*Back*", al encontrarse en alguna de las gráficas. 2a. En el caso de haberse iniciado la aplicación, el *splash screen* no se muestra.		

Tabla 47. CU–04 / Modificar parámetros del sistema

ID: CU–05			
Título:	Salir de la aplicación	**Actor:**	Usuario
Descripción:	El usuario decide salir de la aplicación para no recibir más activaciones de sensores.		
Precondiciones:	El sistema ha de haber sido iniciado.		
Postcondiciones:	Se cierra la interfaz de usuario, se liberan recursos (referencia a base de datos, *listeners*, etc.) y se detiene el servicio encargado de recibir activaciones.		
Escenario Principal:	1. El usuario accede a la aplicación. 2. Hace clic en el *Splash Screen* o espera 5 segundos. 3. Pulsa el botón de "Opciones" del dispositivo móvil. 4. El usuario elige la opción de "*Stop service & exit*". 5. La aplicación se cierra.		
Escenario Alternativo:	1a. El usuario puede ya encontrarse en la propia aplicación. En cuyo caso, deberá pulsar la tecla "*Back*", al encontrarse en las Preferencias o en alguna de las gráficas. 2a. En el caso de haberse iniciado la aplicación, el *splash screen* no se muestra.		

Tabla 48. CU–05 / Salir de la aplicación

Tras describir los casos de uso de la aplicación se procede a mostrar los distintos diagrama de casos de uso, los cuales se pueden observar en la Figura 28, en la Figura 29 y en la Figura 30. Los casos de uso se representan por elipses y se muestran como parte del contexto o aplicación que está siendo modelado, los actores sin embargo se externalizan.

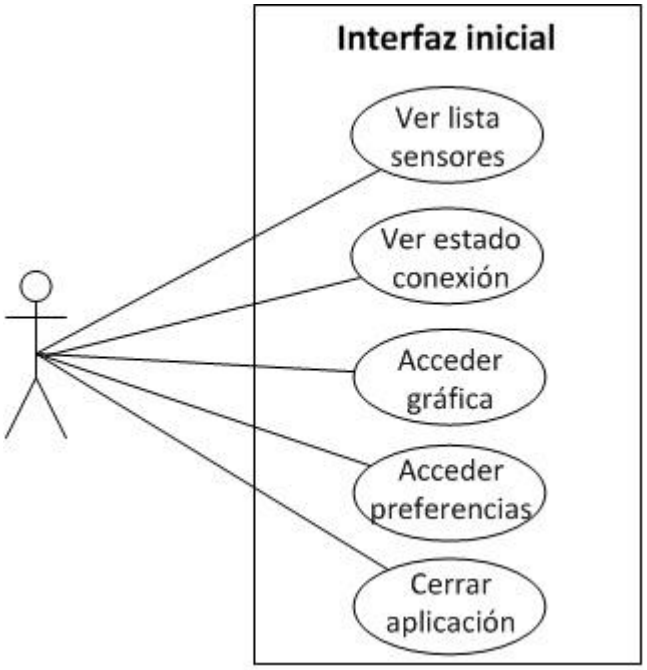


Figura 28. Diagrama de casos de uso – Interfaz inicial

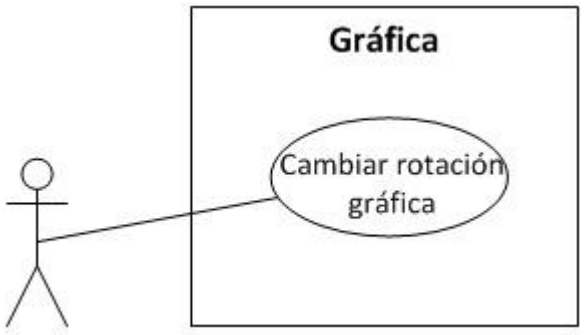


Figura 29. Diagrama de casos de uso – Gráfica

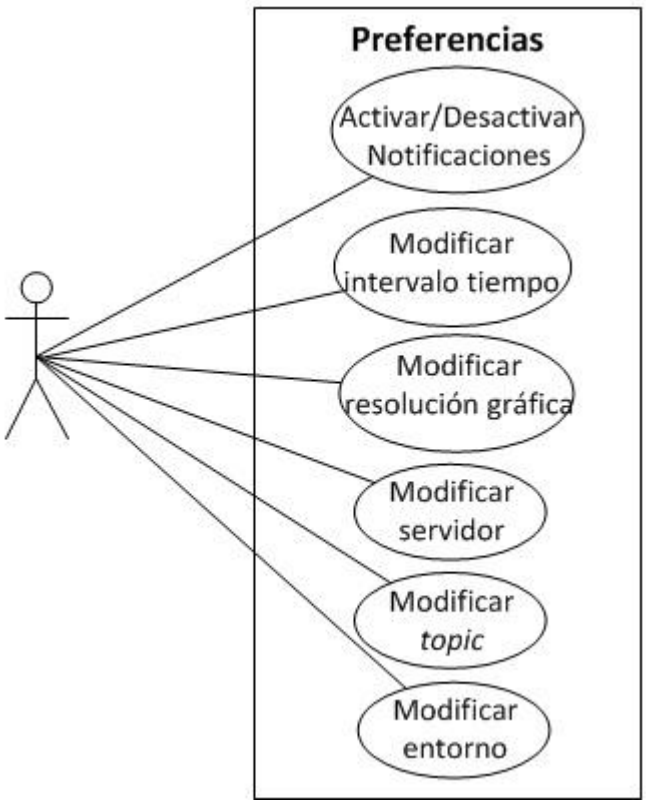


Figura 30. Diagrama de casos de uso – Preferencias

5.2 Diseño

El objetivo de esta fase es la de modelar el sistema basándose en la salida de la fase anterior (análisis), para que posteriormente el personal encargado de programación, pueda implementar el proyecto. La etapa de diseño permite a su vez tener una idea más clara y "física" de cómo será el sistema, ya que quita parte de abstracción de la etapa anterior.

5.2.1 Arquitectura de la aplicación

Ya se comentó en el apartado 2.5 "Diseño arquitectónico y detallado", que el patrón arquitectónico a emplear por esta aplicación es el llamado Modelo Vista Controlador, o comúnmente designado por sus siglas MVC. El mismo nombre del patrón indica la separación en tres componentes muy distintos, relacionados entre sí, de forma que se permita separar los datos de la aplicación, de su interfaz y de su lógica de negocio.

Este tipo de arquitecturas permiten por un lado dividir al equipo de trabajo más fácilmente, y por otro desarrollar componentes o librerías que puedan ser reutilizadas en un futuro por algo más que el proyecto actual.

Los componentes del modelo MVC se relacionan tal y como indica la Figura 31.

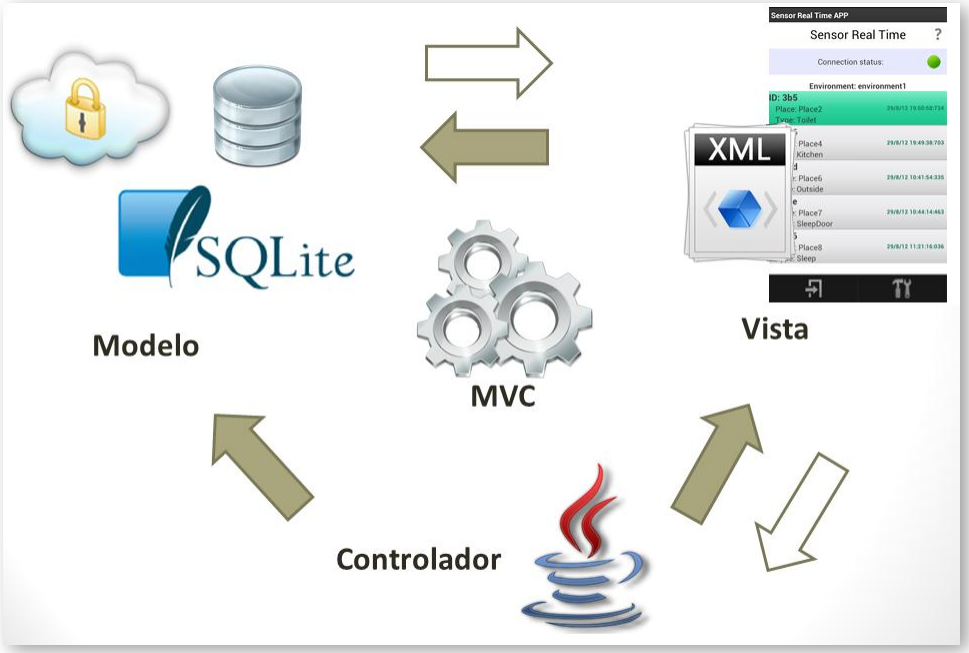


Figura 31. Modelo MVC

Cada uno de estos tres componentes se detalla a continuación:

- Modelo:
  Este componente hace referencia a las representaciones creadas en base a la información con la que tratará la aplicación. Gracias a esto se intentará cumplir el principio de desarrollo software *DRY* o "*Don´t Repeat Yourself*" [56], intentando siempre reducir cualquier tipo de información redundante. De esta forma cada elemento de conocimiento (en este caso en concreto, la noción de sensor, por ejemplo) debe tener una única e inequívoca representación dentro del sistema. Así pues una modificación en un único elemento del sistema, no implica un cambio en otras partes lógicas que no estén relacionadas con este último. En el modelo también se tendrá en cuenta el empleo de una base de datos *SQLite* para almacenar información.

- Vista:
  La vista es la parte con la que puede interactuar el usuario, es decir, la interfaz de usuario o *UI*. Dichas interfaces están construidas sobre ficheros *XML*, y pueden separar la estructura del estilo tal y como lo hacen *HTML* y *CSS*.

- Controlador:
  Por un lado tenemos los datos, por otro las interfaces y por último necesitamos del controlador que será el encargado de gestionar la lógica de la aplicación, de responder a eventos, peticiones del usuario, etc. El controlador estará formado por clases programadas en lenguaje *Java*, pudiéndose entender como el *core* de la aplicación.

Gracias a este tipo de modelos se puede obtener una aplicación escalable, construida mediante perfiles especializados que se encargarán de desarrollar cada uno de los componentes. De esta forma el diseñador gráfico no necesitará más que saber *XML* desentendiéndose de la lógica de negocio, y lo mismo ocurrirá con el encargado del modelado de datos y el del controlador.

Un flujo clásico de esta arquitectura podría ser el siguiente:

1. El usuario comienza interactuando con la aplicación. El encargado es la vista (El usuario quiere ver la gráfica de un sensor).
2. El controlador recibe dicha acción (A través de un *handler* comprobará si ha clicado en un elemento de la lista de sensores).
3. El modelo es llamado para acceder a las últimas activaciones de dicho sensor.
4. Una vez obtenidas todas las activaciones, el controlador tomará partida para llamar a la vista y renderizar la gráfica.
5. El usuario obtendría la nueva interfaz con la gráfica de las últimas activaciones del sensor que eligió. El ciclo volvería a iniciarse cuando se solicitase una nueva acción.

5.2.2 Modelado del sistema

El modelado de un sistema sirve como vista gráfica de la información que se especificó con anterioridad en la fase de requisitos. Se podría decir que es una abstracción o simplificación del sistema real a desarrollar, que sirve como es obvio para poder comprenderlo mejor. Dicho modelo estará formado por distintos diagramas o vistas parciales del modelo que servirán como base para la futura implementación.

Cabe destacar que debido a la escasa complejidad del proyecto, se han obviado los diagramas de actividad y los casos de uso reales, al no aportar más información acerca de la funcionalidad que la que ofrecían los propios casos de uso de la fase de análisis.

A lo largo de esta sección se mostrarán los distintos tipos de diagramas empleados para modelar el sistema entre los que se encuentran: diagrama de flujo, diagrama de secuencia, diagrama de clases, diagrama estructural y diagrama de navegación. Por último se mostrará la trazabilidad entre los requisitos software y los componentes de la arquitectura.

5.2.2.1 Diagramas de flujo

El diagrama de flujo sirve para representar de forma gráfica el proceso del sistema, es decir, mediante símbolos con significados definidos e interconectados con flechas se marcan los pasos del sistema. Dicho diagrama se caracteriza por tener un único punto de inicio y un único punto de finalización.

Su simbología es la siguiente:

- Óvalo/Elipse: Inicio y fin
- Rectángulo: Actividad
- Rombo: Decisión
- Paralelogramo inclinado: Datos / Almacenamiento

Se mostrará un diagrama de flujo por cada una de las posibles acciones básicas o procesos que se puede desempeñar dentro del sistema.

En la Figura 32 se muestra el diagrama de flujo correspondiente a la acción de acceder a la lista de sensores activados.

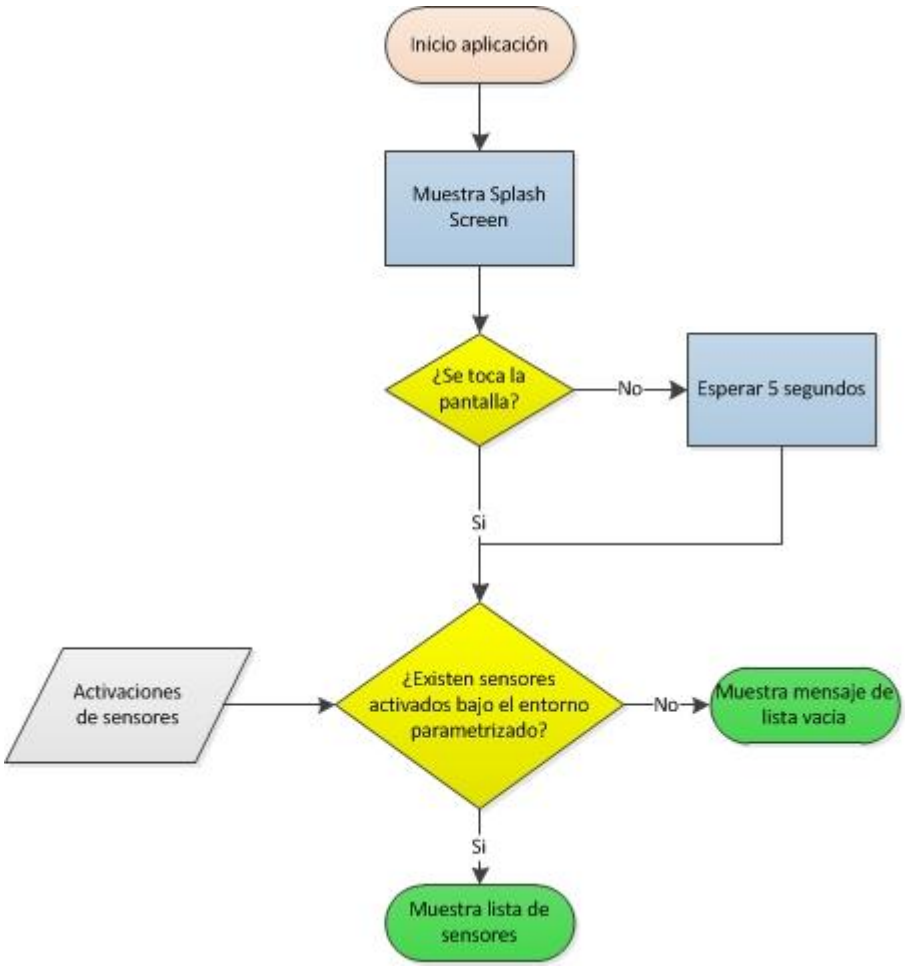


Figura 32. Diagrama de flujo – Acceder lista de sensores

En la Figura 33 se muestra el diagrama de flujo correspondiente a la acción de obtener estado de la conexión.

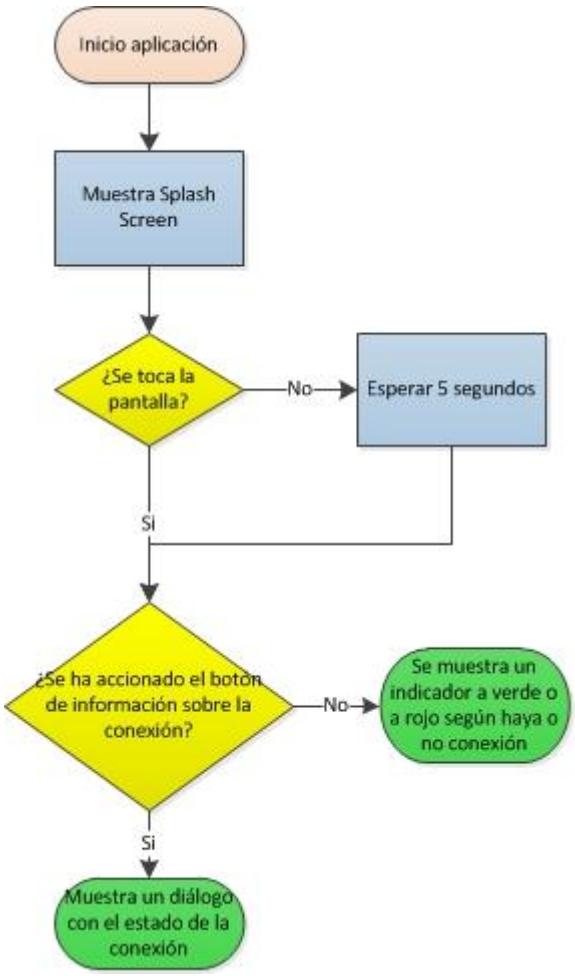


Figura 33. Diagrama de flujo – Obtener estado conexión

En la Figura 34 se muestra el diagrama de flujo correspondiente a la acción de mostrar la gráfica de activaciones de un determinado sensor.

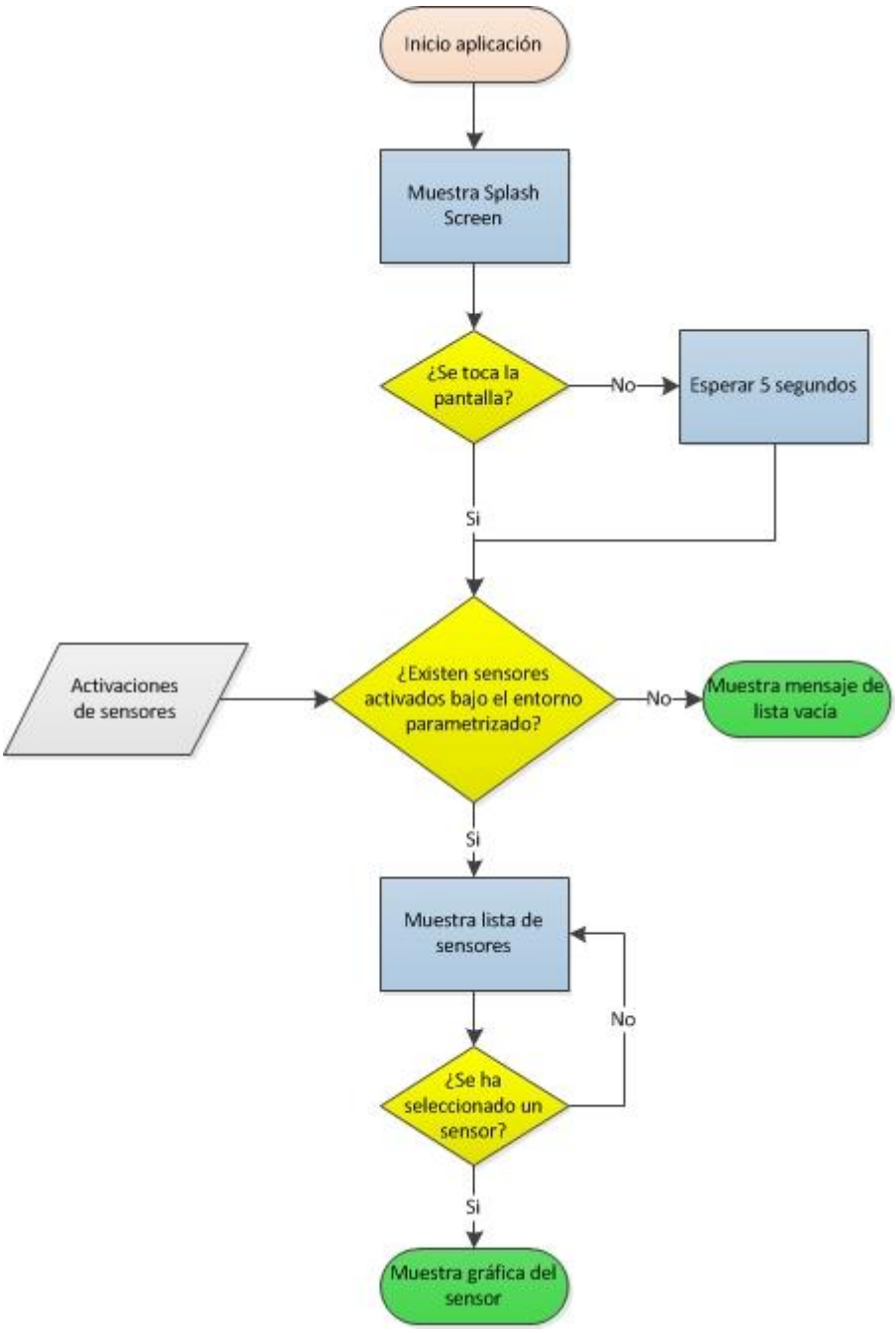


Figura 34. Diagrama de flujo – Mostrar gráfica sensor

En la Figura 35 se muestra el diagrama de flujo correspondiente a la acción de modificar parámetros del sistema.

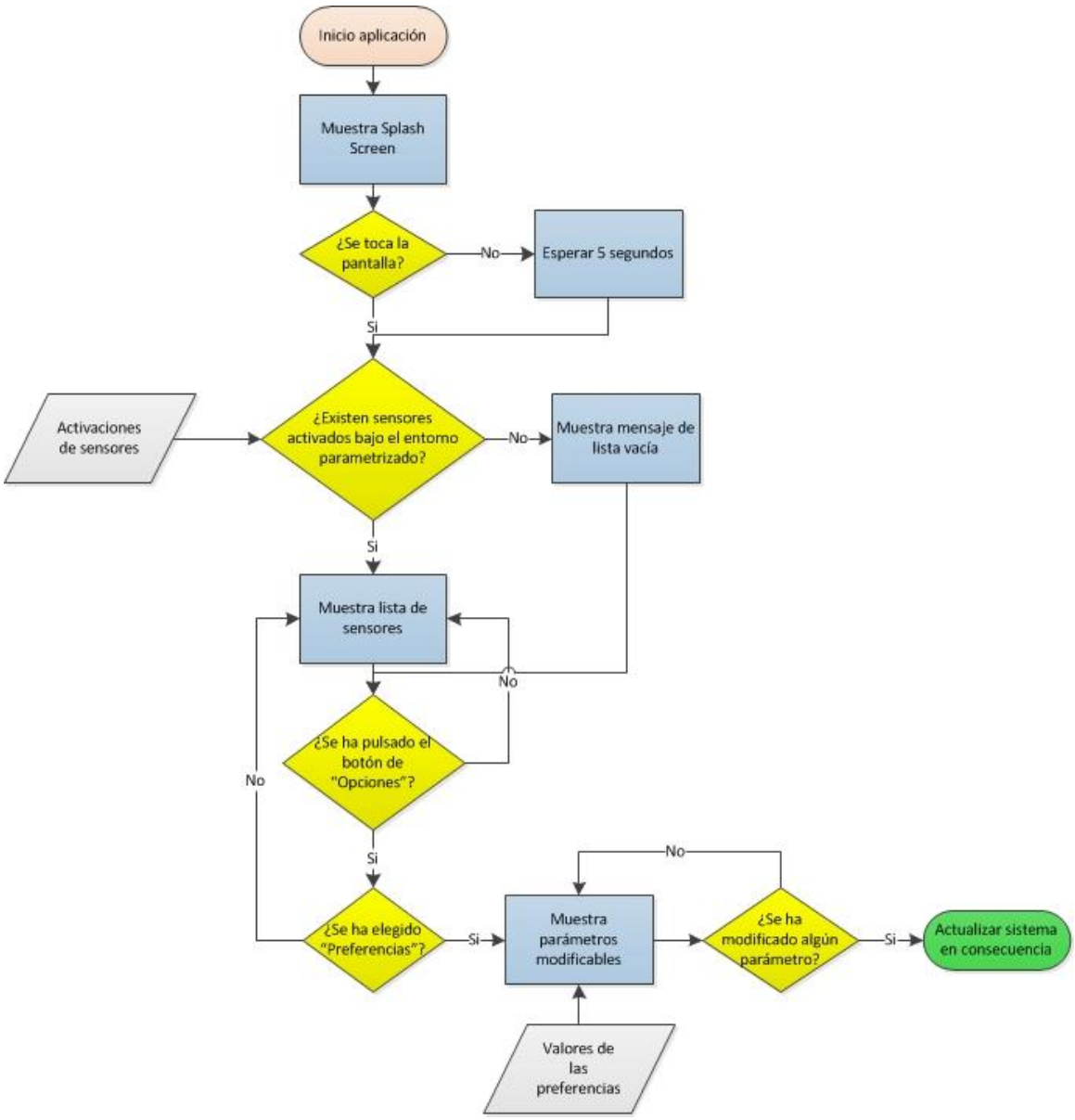


Figura 35. Diagrama de flujo – Modificar parámetros preferencias

En la Figura 36 se muestra el diagrama de flujo correspondiente a la acción de salir del sistema.

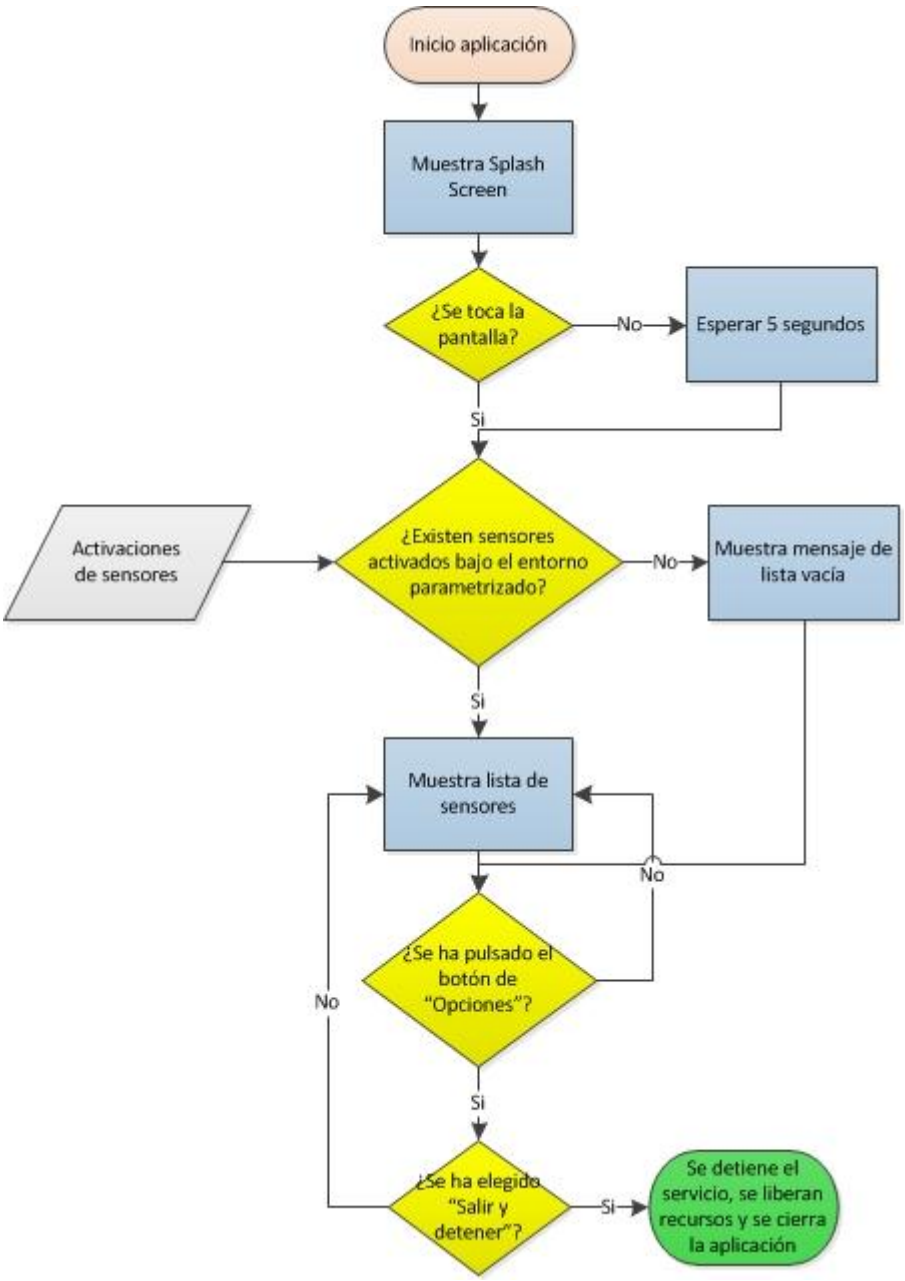


Figura 36. Diagrama de flujo – Cerrar aplicación

En la Figura 37 se muestra el diagrama de flujo correspondiente a la recepción de una activación de sensor.

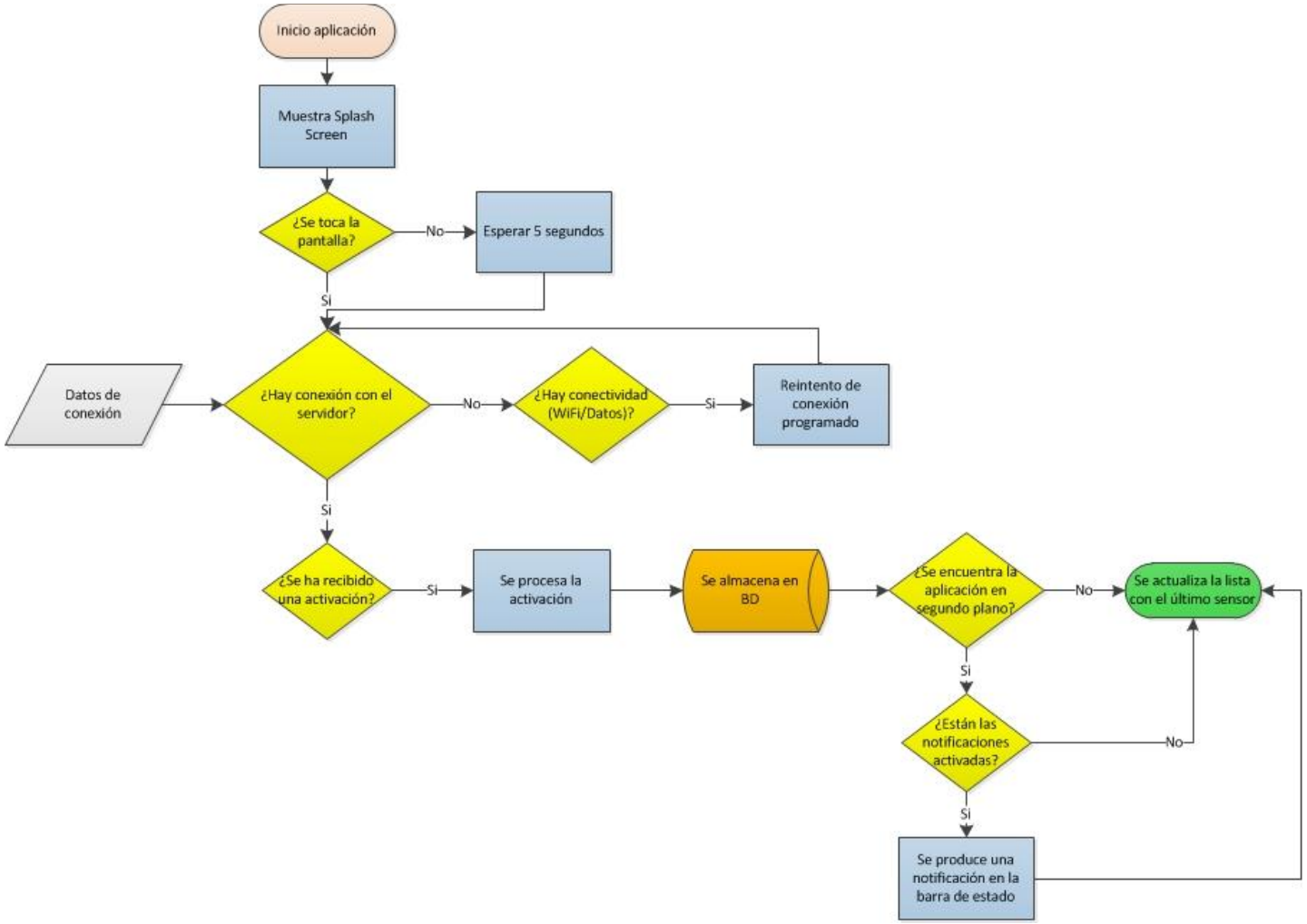


Figura 37. Diagrama de flujo – Recepción de activación

5.2.2.2 Diagrama de clases

Este tipo de diagramas son diagramas estáticos que describen la estructura del sistema, mostrando las clases, con sus atributos y métodos correspondientes, y con las relaciones entre ellas. El diagrama de clases del proyecto a desarrollar se puede apreciar en la Figura 38.

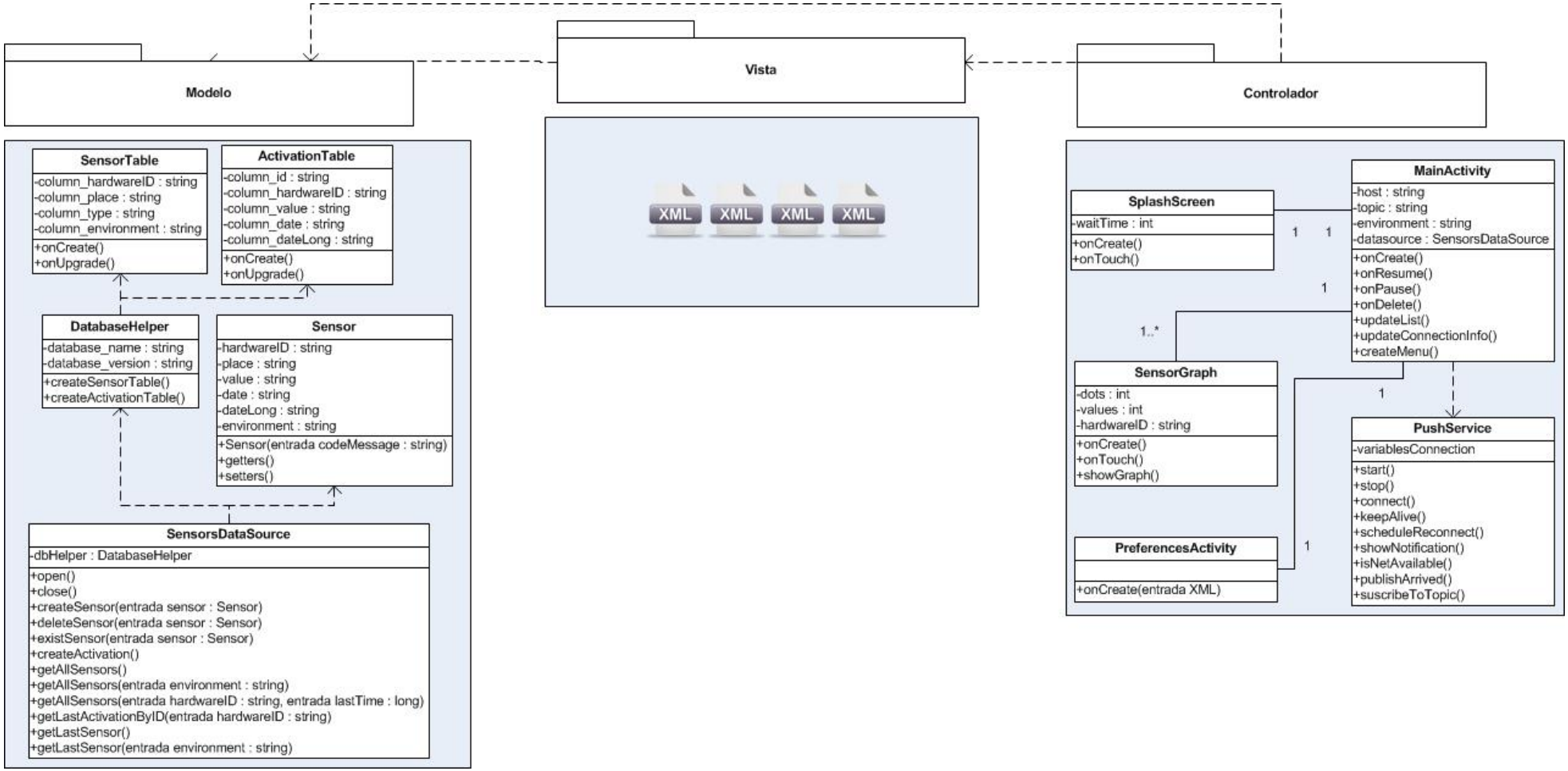


Figura 38. Diagrama de clases

El componente referente a la Vista, como es obvio, no se refleja dentro del diagrama de clases al estar formado únicamente por ficheros de formato .xml. Al no estar formado dicho componente por ninguna clase, no se puede mostrar una relación directa entre el resto, ni en forma de dependencias ni de asociaciones.

A continuación se explica brevemente tanto el componente del Modelo como el del Controlador para clarificar cualquier posible duda que pueda surgir tras observar el diagrama:

- Modelo

  En el componente referente al Modelo se pueden distinguir dos partes, por un lado los objetos con los que trabajará el Controlador, en este caso objetos de tipo "*Sensor*" y por otro la capa de persistencia, que en este caso estará formada por dos tablas: "*SensorTable*" y "*ActivationTable*".

  Para poder ofrecer extensibilidad, lo óptimo y más eficiente suele ser ofrecer una clase por cada tabla que se requiera en la base de datos. De esta forma en la tabla "*SensorTable*" se almacenarían los sensores que se hayan ido activando, guardando un registro independientemente del número de veces que éste se haya activado. En la tabla "*ActivationTable*" por el contrario, sí que se almacenarían todas las activaciones (con referencia al sensor de la primera tabla), para disponer de suficiente información con la que dibujar las gráficas en un futuro. Todas las tablas que requiera la aplicación las instancia la clase "*DatabaseHelper*" nada más crear un objeto de este tipo.

  Con el fin de poder convertir los datos de la capa de persistencia a objetos manipulables por el sistema, existe la clase "*SensorsDataSource*" que actuará de intermediaria a modo de *DAO* (*Data Acces Object* [57]). Gracias a esta clase se realizarán las distintas consultas e inserciones en la base de datos.

- Controlador

  En cuanto al Controlador; está formado en una gran parte por actividades y servicios. Estos son dos conceptos a la hora de diseñar en Android que deben de quedar claros de antemano. Los servicios son empleados frecuentemente para realizar tareas en segundo plano durante un periodo de tiempo continuado y sin necesidad de mostrar elementos visuales, mientras que lo lógico es pensar en las actividades como "pantallas visuales".

  Así pues la clase "*PushService*" es un servicio dedicado a la conexión mediante *MQTT* al servidor que corresponda y a la suscripción del *topic* especificado. La clase "*MainActivity*" es la actividad encargada de comenzar el servicio "*PushService*" y de mostrar la lista de sensores, la información sobre el estado de la conexión, y la posibilidad mediante la barra de menú de acceder a las preferencias o salir de la aplicación. Al pinchar en uno de los sensores de la lista se realizaría una llamada a la clase "*SensorGraph*", encargada de dibujar la gráfica correspondiente a las activaciones del sensor. La actividad "*PreferencesActivity*" se limita a cargar un archivo de preferencias .xml, y la "*SplashScreen*" una pantalla de bienvenida (tras cierto tiempo o al tocar la pantalla se cargaría "*MainActivity*").

5.2.2.3 Diagramas de secuencia

A pesar de que ya con los diagramas de flujo y los diagramas de clase se puede tener un 75% de los atributos de las clases identificados, no es hasta los diagramas de secuencia donde se comienza a ver con claridad la relación entre clases y la interactuación mediante métodos y mensajes. Así pues, en un diagrama de este tipo se indican las clases o módulos que forman parte de la aplicación y las llamadas que se hacen en cada uno de ellos para desempeñar una tarea concreta.

A continuación se mostrará un diagrama de secuencia por cada uno de los casos de uso que se especificaron en el apartado de análisis:

- CU–01: Obtener la información listada de los sensores activados hasta el momento
- CU–02: Obtener estado de la conexión
- CU–03: Mostrar gráfica de activaciones en función del sensor elegido
- CU–04: Modificar parámetros principales del sistema
- CU–05: Salir de la aplicación

Estos diagramas se pueden apreciar correspondientemente en la Figura 39, en la Figura 40, en la Figura 41, en la Figura 42 y en la Figura 43.

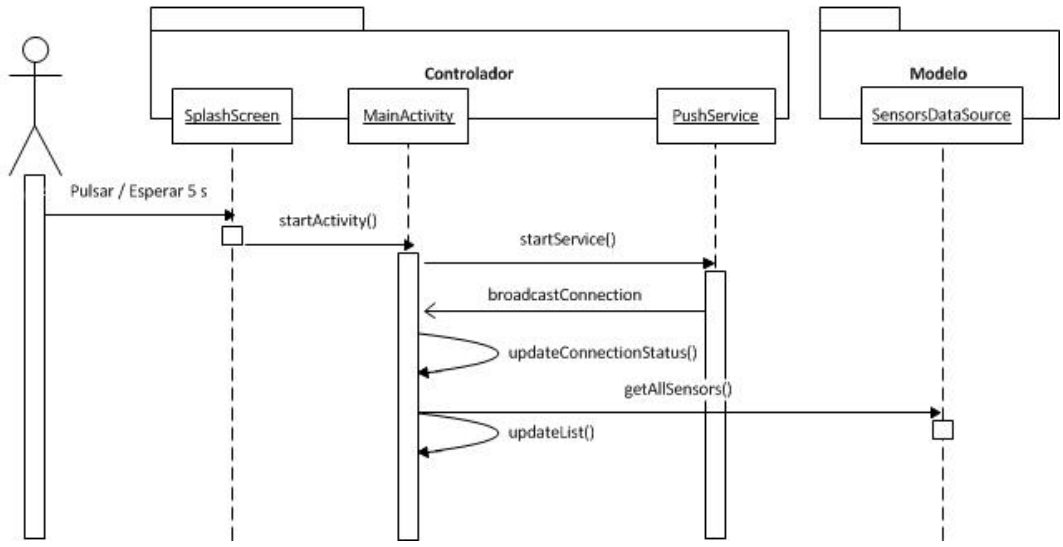


Figura 39. Diagrama de secuencia – Obtener lista de sensores

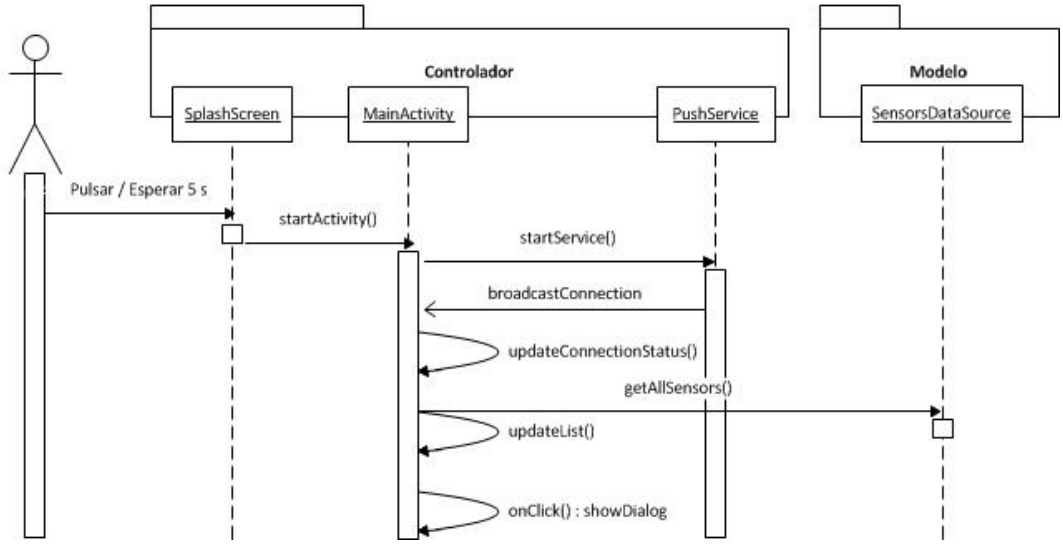


Figura 40. Diagrama de secuencia – Obtener estado de conexión

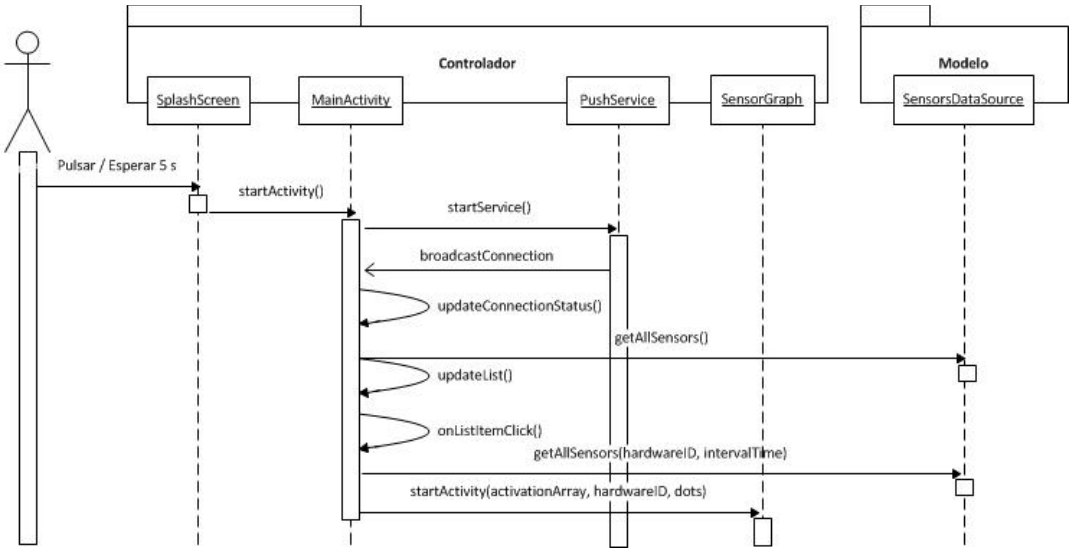


Figura 41. Diagrama de secuencia – Mostrar gráfica de activaciones

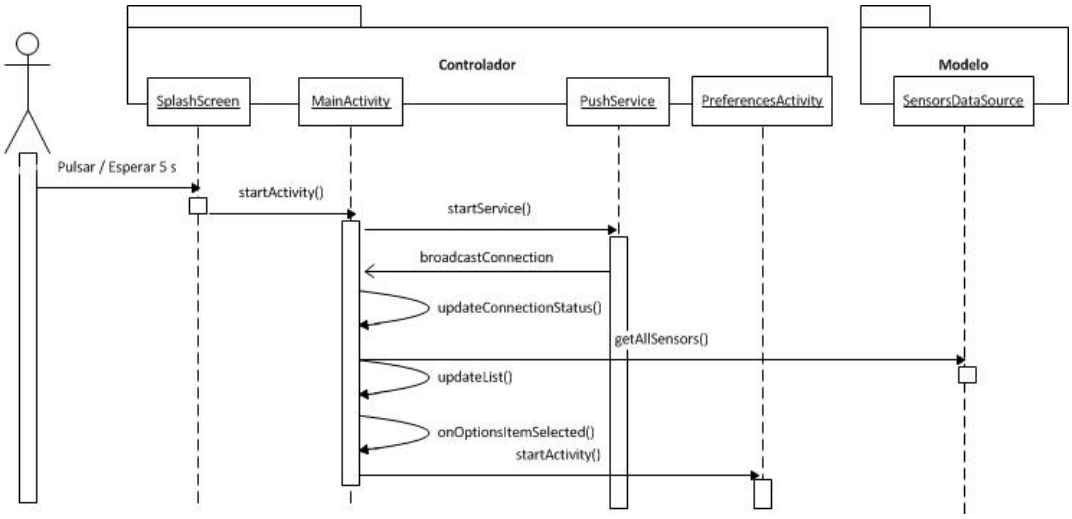


Figura 42. Diagrama de secuencia – Modificar parámetros del sistema

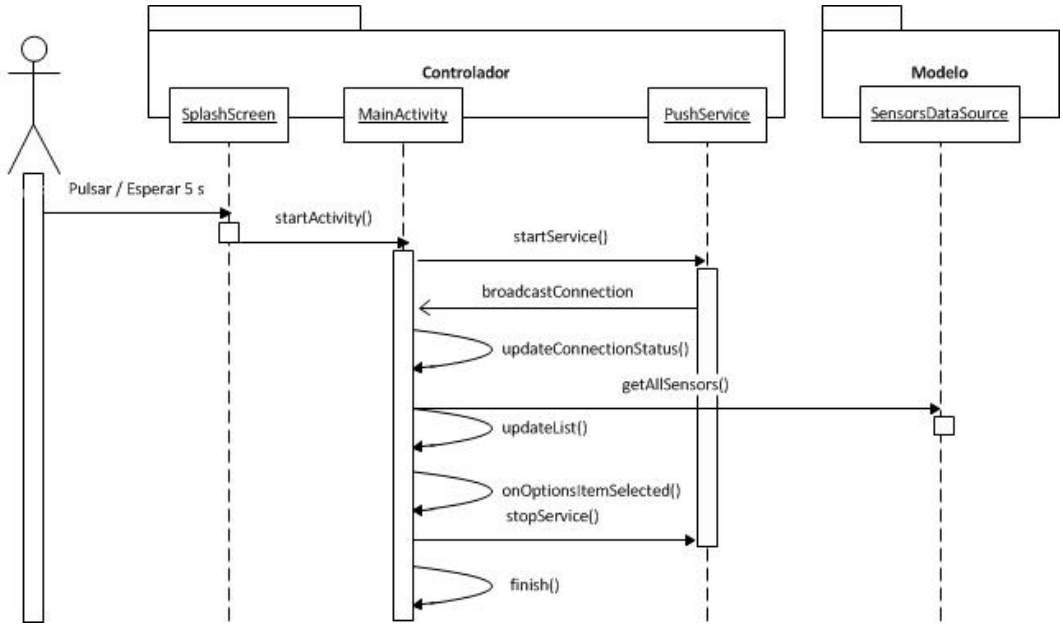


Figura 43. Diagrama de secuencia – Salir de la aplicación

5.2.2.4 Diagrama estructural

A continuación se incluye en la Figura 44 el diagrama estructural con el fin de identificar y representar los componentes del sistema y las relaciones estructurales entre ellos. Los nodos de información pueden ser simples o compuestos.

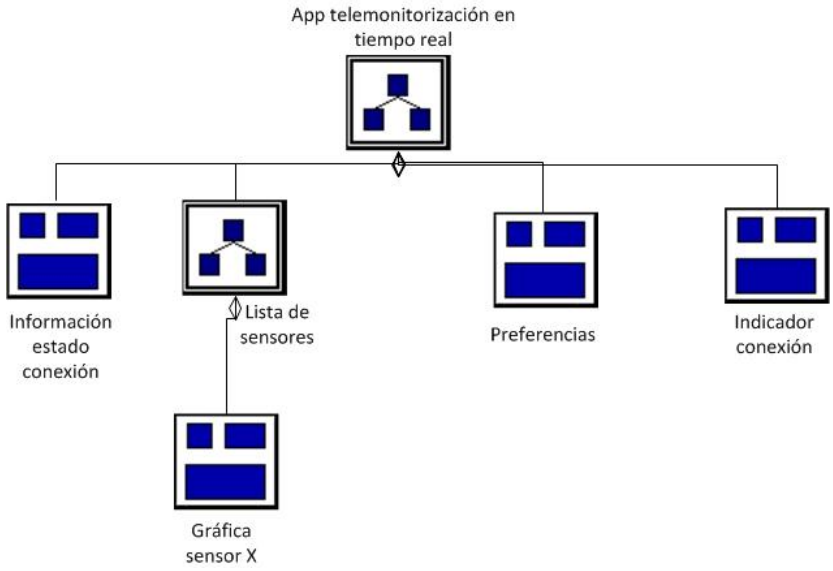


Figura 44. Diagrama estructural

5.2.2.5 Diagrama de navegación

Tras definir el diagrama estructural, a continuación se presenta en la Figura 45 el diagrama de navegación especificando las rutas existentes entre los nodos del sistema definidos. Este tipo de diagramas está conformado por dos elementos esenciales: los nodos de información (definidos por el diagrama estructural) y los enlaces entre ellos. Dichos enlaces pueden ser unidireccionales (color azul) o bidireccionales (color rojo).

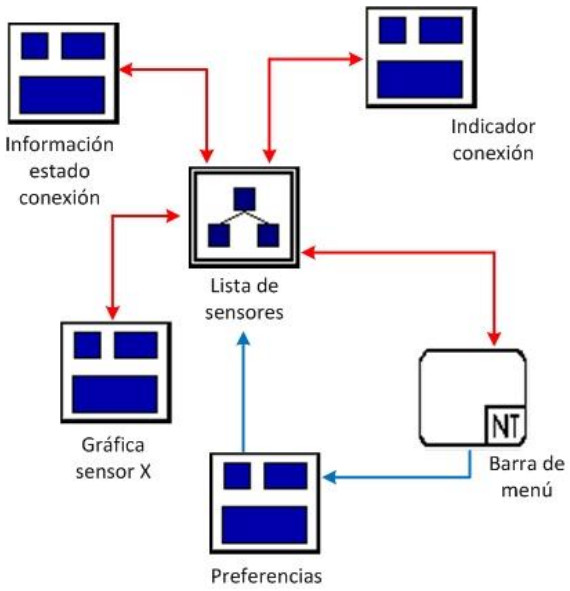


Figura 45. Diagrama de navegación

5.2.2.6 Trazabilidad: Requisitos –> Diseño

Por último queda comprobar que todo requisito software se encuentre al menos en un componente de la arquitectura. Los componentes a verificar son el Modelo, la Vista y el Controlador, tal y como especifica la arquitectura del modelo MVC que se está empleando. Dicha comprobación se puede observar en la Tabla 49.

	MODELO	VISTA	CONTROLADOR
RS–F01	X		
RS–F02	X	X	X
RS–F03	X		
RS–F04	X	X	X
RS–F05			X
RS–F06		X	X
RS–F07		X	X
RS–F08	X	X	X
RS–F09	X	X	X
RS–F10			X
RS–F11		X	X
RS–F12		X	X
RS–F13	X	X	X
RS–F14	X	X	X
RS–F15	X		X
RS–NF01			X
RS–NF02			X
RS–NF03	X		
RS–NF04			X
RS–NF05		X	
RS–NF06		X	
RS–NF07			X

Tabla 49. Matriz de trazabilidad RS/Componentes

Una vez establecido el diseño y comprobado que mantiene lo demandado por el cliente sería el momento para comenzar con la implementación del sistema.

5.3 Implementación

Una vez realizado el diseño se comenzaría con la parte a más bajo nivel de la aplicación, es decir, es el momento de implementarla empleando el entorno y el lenguaje adecuado. En esta sección se comentarán tanto las distintas decisiones de implementación empleadas como los problemas encontrados.

5.3.1 Decisiones de implementación

A continuación se desglosan las distintas decisiones empleadas:

5.3.1.1 Lenguaje, entorno y estructura del proyecto

Salvo para la parte gráfica de las vistas que se programan en *XML*, el lenguaje destinado a la creación de clases realmente no se puede elegir, ya que todas las aplicaciones Android se crean sobre el lenguaje Java. Cabe destacar que archivos y librerías propias de este SO están implementadas en C/C++, pero ese es otro tema.

En lo referente al IDE se podría haber elegido tanto *NetBeans* como *Eclipse*, ya que ambos permiten el desarrollo en lenguaje Java, pero finalmente se decidió emplear este último, ya que el kit de desarrollo de Android contiene un *plugin* para *Eclipse*. Esto permite preocuparse únicamente de la codificación del sistema, al encargarse el *plugin* de la configuración del proyecto, permitiendo compilar y ejecutar como si fuera un proyecto en Java común y corriente. Se ofrece además la posibilidad de compilar y "*debugear*" tanto sobre un emulador Android como directamente sobre un dispositivo conectado por *USB*. Para realizar este proyecto se decidió seguir la segunda opción por su eficiencia y rapidez.

En la Figura 46 se muestra la estructura del proyecto dentro del entorno *Eclipse*.

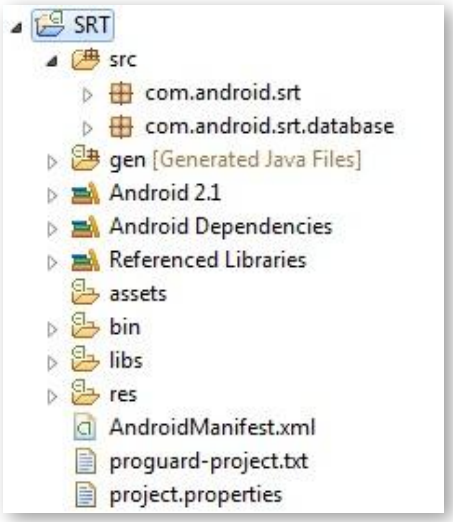


Figura 46. Estructura del proyecto

Respecto a las clases se decidió separar las relacionadas con la base de datos en un paquete separado del resto, tal y como se ve en la Figura 47, con el fin de modular el código según responsabilidades tanto como fuera posible.

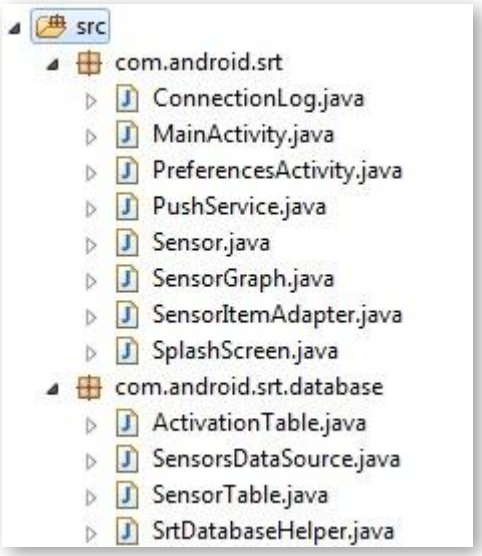


Figura 47. Estructura del proyecto – Source

Con el fin de que los recursos gráficos puedan adaptarse correctamente a las distintas resoluciones de pantalla, se ha estructurado dicho directorio de forma que hay recursos específicos para cada una de las resoluciones [58]: l*dpi* (*low*), m*dpi* (*medium*), *hdpi* (*high*), *xhdpi* (*extra high*), tal y como se muestra en la Figura 48.

Figura 48. Estructura del proyecto – Resources

A su vez dentro del mismo directorio de "*Resources*" se ha decidido separar los colores y las cadenas de texto de las distintas interfaces gráficas, tal y como se aprecia en la Figura 49.

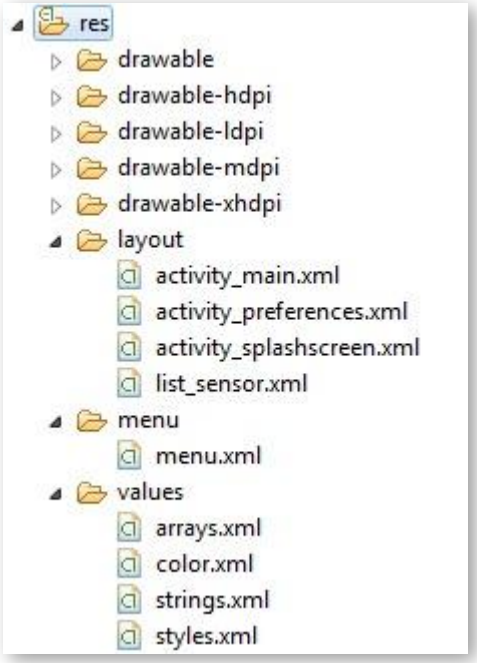


Figura 49. Estructura del proyecto – Resources 2

Se ha decidido también crear un directorio para almacenar todas las distintas librerías de las que requiera el proyecto para funcionar con normalidad, como muestra la Figura 50. En dicha figura se muestran tres librerías concretamente; una es necesaria para el soporte de clases con nivel de API 4 o posterior [59], la de "*wmqtt*" es una implementación simple del protocolo *MQTT* aportada por IBM para hacer uso de métodos relevantes y "*achartengine*" que es una librería de código abierto dedicada al uso de gráficas en Android. Se ha decidido emplear una librería para dibujar las gráficas, ya que posee muy buena adaptación, es relativamente sencilla de implementar, es de licencia libre y porque de lo contrario habría que haber empleado "*Drawables*" en Android para emular este mismo cometido, lo cual lo haría bastante engorroso.

Figura 50. Estructura del proyecto - Libraries

5.3.1.2 Base de datos

Como se comentó con anterioridad la base de datos empleada ha sido *SQLite*. *SQLite* es una base de datos *open source* integrada en Android. Dicha base de datos soporta características estándar relacionales como sintaxis *SQL*, transacciones y "*prepared statements*". Además cabe resaltar que ocupa muy poca memoria durante tiempo de ejecución (aproximadamente 250 KBytes).

Por ocupar lo poco que ocupa, *SQLite* soporta únicamente datos de tipo *TEXT*, *INTEGER*, y *REAL*, los cuales serían similares al *String*, al *long* y al *double* de Java respectivamente. Cualquier otro tipo de dato debe convertirse a uno de los anteriores antes de ser almacenado en la base de datos.

En dicha base de datos la aplicación a desarrollar necesita llevar un registro de las activaciones que ha tenido cada sensor desde que la aplicación se lanzó por primera vez (en caso de que no se hubiesen borrado los datos manualmente). Para ello se decidió crear dos tablas, tal y como indica el sencillo modelo relacional de la Figura 51.

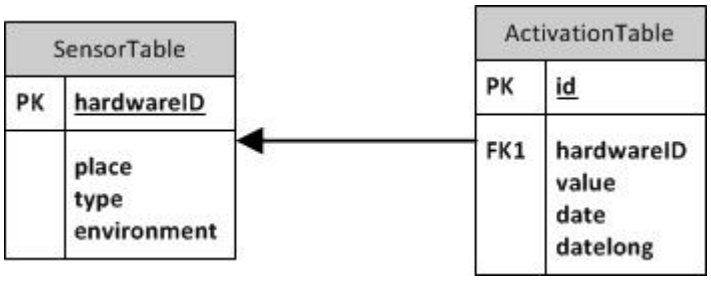


Figura 51. Modelo relacional BD

De esta forma con cada activación recibida, insertaríamos los datos básicos del sensor en la primera tabla (comprobando previamente que no exista ya un registro con esa clave *hardware*), y posteriormente se almacenaría la activación correspondiente guardando el valor obtenido (recordar que son sensores binarios, 0 o 1) y la fecha en la que se activó. En cuanto a la fecha, se guarda tanto el *string* como el formato en long, ya que el formato numérico nos permitirá realizar operaciones y filtrar, por ejemplo, sensores que hayan sido activados dentro de un cierto intervalo de tiempo.[7] Permitiendo filtrar directamente a la hora de hacer la *query* o petición a base de datos, en lugar de descargar todos los registros para posteriormente realizar las conversiones y descartar elementos.

[7] Una fecha en formato *long* hace alusión al número de milisegundos transcurridos desde el 1 de Enero de 1970, por tanto, una activación más tardía tendrá un número mayor.

5.3.2 Problemas encontrados

A continuación se resumen brevemente algunos de los problemas que se encontraron durante la implementación final del proyecto:

- Nivel API Android:

  Al comienzo de la implementación se intentó hacer funcionar la conexión mediante la librería de *MQTT* para un nivel del API de 15, sin mucho éxito. Cada nivel de API está soportado por una o ciertas versiones de la plataforma Android, de esta forma, la API 15 corresponde a la 4.0.3 y a la 4.0.4 del sistema operativo. Fue de interés programar para esta versión debido a las nuevas funcionalidades que eso implicaba y a la homogeneización de estilos con el reciente tema *Holo* [60], aun dejando fuera a muchos dispositivos que no pudieran ejecutar un código tan moderno. Finalmente no se consiguió hacer funcionar dicha librería junto con ese API, y se decidió aumentar el rango de dispositivos programando para el nivel 7, es decir, cualquier dispositivo con SO superior al 2.1.x (o código de versión "*ECLAIR_MR1*").

- *Life cycle* de Android:

  El ciclo de vida de las actividades de Android supuso al comienzo un pequeño quebradero de cabeza, ya que la respuesta de la actividad a un inicio desde 0, a un inicio por estar en segundo plano o por ejemplo al volver de otra actividad es completamente distinta.

  Desde el punto de vista del proyecto, la idea era que existiese un servicio de fondo que se encargase de la conexión (recibir eventos, reconectarse, avisar en caso de desconexión, etc.) y que no se terminase aun saliendo de la aplicación. Un ejemplo parecido es el de un reproductor de música que al dejar la aplicación en segundo plano, la música no deja de reproducirse. Bien es cierto que en caso de desear finalizar la aplicación (en este caso desde la actividad principal "*MainActivity*"), la ejecución del servicio dejaría de tener sentido y habría que detenerla. Esta acción junto con la de liberar otro tipo de recursos (*listeners*, referencia a base de datos, etc.) se realiza dentro del método *onDestroy()*, llamado justo antes de finalizar la vida de la app. Uno de los problemas, por ejemplo, surgía a la hora de rotar la pantalla, ya que el SO se encargaba de forma automática de finalizar la actividad y volverla a crear, lo cual suponía un problema tal y como se había planteado el ciclo de vida.

6 Pruebas y evaluación

"Por norma, los sistemas software no funcionan bien hasta que han sido utilizados y han fallado repetidamente en entornos reales".

Dave Parnas

En este capítulo se presentan las pruebas realizadas al sistema, con las cuales se aseguran que los requisitos especificados se han cumplido. Se describe el entorno de pruebas, y a continuación se detallan las mismas. En el último punto de este apartado se hará un pequeño resumen de los resultados obtenidos.

6.1 Descripción del entorno de pruebas

Se ha decidido emplear tres dispositivos Android de última generación en la realización de las pruebas. Por un lado se ha dispuesto de dos *smartphones* y por otro de un *Tablet*, con las siguientes características:

- Sony Ericsson Xperia Neo V
  - SO: Android v4.0.3 (*Ice Cream Sandwich*)
  - CPU: 1 GHz Scorpion
  - GPU: Adreno 205
  - GPS: Si, con soporte A-GPS
  - Java: Si, vía emulador Java *MIDP*
  - Dimensiones: 116 x 57 x 13 mm
  - Peso: 126 g
  - *Display*: 480x854 *pixels* (~265 ppi *pixel density*)
  - Multitouch: Si

Figura 52. Dispositivo pruebas 1 – Sony Ericsson Xperia Neo V

- Samsung I9100 Galaxy S II
  - SO: Android v2.3.4 (*Gingerbread*)
  - CPU: *Dual-core* 1.2 GHz Cortex-A9
  - GPU: Mali-400MP
  - GPS: Si, con soporte A-GPS
  - Java: Si, vía emulador Java *MIDP*
  - Dimensiones: 125.3 x 66.1 x 8.5 mm
  - Peso: 116 g
  - *Display*: 480x800 *pixels* (~217 ppi *pixel density*)
  - Multitouch: Si

Figura 53. Dispositivo pruebas 2 – Samsung Galaxy SII

- Asus Transformer TF101
  - SO: Android v4.0.3 (*Ice Cream Sandwich*)
  - CPU: *Dual-core* 1 GHz Cortex-A9
  - GPU: ULP GeForce
  - GPS: Si, con soporte A-GPS
  - Java: Si, vía emulador Java *MIDP*
  - Dimensiones: 271 x 171 x 13 mm
  - Peso: 680 g
  - *Display*: 1280x800 *pixels* (~149 ppi *pixel density*)

Figura 54. Dispositivo pruebas 3 – Samsung

6.2 Validación realizada

A continuación se muestran las pruebas realizadas sobre el sistema para comprobar su correcto funcionamiento, tanto a nivel de código como comprobando el sistema en ejecución.

6.2.1 Casos de prueba

Los *Test Case* o casos de prueba son un conjunto de variables o condiciones gracias a las cuáles podrá determinar el analista si lo especificado en el apartado del análisis es parcial o completamente satisfactorio. Para ello se ha decidido realizar un caso de prueba por cada uno de los casos de uso expuestos en el apartado 5.1.2. A continuación se muestran los casos de prueba:

ID: CP–01	
Propósito:	Se está probando el aspecto del sistema referente a mostrar la lista de sensores.
Prerrequisitos:	No existen prerrequisitos.
Datos de Prueba:	Lista de variables y sus posibles valores usados en el caso de prueba: listaSensores = {lista vacía, lista con al menos un objeto Sensor}
Pasos a ejecutar la prueba:	1. Acceder a la aplicación. 2. Hacer clic en el *Splash Screen* o esperar 5 segundos. 3. Comprobar que en caso de haberse registrado sensores deberían mostrarse en la lista, comprobar en caso contrario que aparece un mensaje de lista vacía.
Notas y Preguntas:	• La lista se filtra por defecto según el entorno elegido en las "Preferencias", por tanto, pueden haberse recibido sensores y aparecer la lista vacía, al no coincidir el entorno de los sensores con el del filtro.

Tabla 50. Caso de Prueba 01 – Mostrar lista de sensores

ID: CP–02	
Propósito:	Se está probando el aspecto del sistema referente a mostrar estado de conexión.
Prerrequisitos:	No existen prerrequisitos.
Datos de Prueba:	Lista de variables y sus posibles valores usados en el caso de prueba: *isConnected*= {*false*, *true*} Host = {*163.117.164.119* , u otra cadena de texto} Topic = {*wfsnrnet/sensor/data*, u otra cadena de texto}
Pasos a ejecutar la prueba:	1. Acceder a la aplicación. 2. Hacer clic en el *Splash Screen* o esperar 5 segundos. 3. Se comprueba que la interfaz muestra una "luz" a verde si isConnected está a true, o roja en a false. 4. En caso de pulsar el botón de información, comprobar que se muestra el host y el topic, coincidiendo con los valores de las preferencias.
Notas y Preguntas:	Ninguna.

Tabla 51. Caso de Prueba 02 – Mostrar estado conexión

ID: CP–03	
Propósito:	Se está probando el aspecto del sistema referente a mostrar la gráfica de sensores.
Prerrequisitos:	No existen prerrequisitos.
Datos de Prueba:	Lista de variables y sus posibles valores usados en el caso de prueba. Al estar limitado por la interfaz, y seleccionar un sensor de una lista existente, al igual que la limitación de la resolución de la gráfica y del intervalo de tiempo, no es necesaria la comprobación de valores que ese escapen de ese dominio: sensor = {sensor seleccionado} dots = {5, 10, 25} interval = {1, 20, 60}
Pasos a ejecutar la prueba:	1. Acceder a la aplicación. 2. Hacer clic en el *Splash Screen* o esperar 5 segundos. 3. Seleccionar el sensor que se desee. 4. Comprobar que las activaciones en la gráfica se corresponden con los registros en la base de datos, según el intervalo elegido.
Notas y Preguntas:	Ninguna.

Tabla 52. Caso de Prueba 03 – Mostrar gráfica sensor

ID: CP–04	
Propósito:	Se está probando el aspecto del sistema referente a modificar parámetros del sistema.
Prerrequisitos:	No existen prerrequisitos.
Datos de Prueba:	Lista de variables y sus posibles valores usados en el caso de prueba: un elemento seleccionado de la lista de preferencias
Pasos a ejecutar la prueba:	1. Acceder a la aplicación. 2. Hacer clic en el *Splash Screen* o esperar 5 segundos. 3. Pulsar botón de "Opciones" y a continuación "Preferencias" 4. Seleccionar un parámetro 5. Modificar parámetro y pulsar "Aceptar" 6. Comprobar al pulsar "Back" que el sistema se ha actualizado en función del parámetro modificado.
Notas y Preguntas:	• El servicio se reinicia en caso de modificar tanto el *host* como el *topic* mediante un *listener*.

Tabla 53. Caso de Prueba 04 – Modificar parámetros del sistema

ID: CP–05	
Propósito:	Se está probando el aspecto del sistema referente a salir del sistema.
Prerrequisitos:	No existen prerrequisitos.
Datos de Prueba:	No hay datos de prueba.
Pasos a ejecutar la prueba:	1. Acceder a la aplicación. 2. Hacer clic en el *Splash Screen* o esperar 5 segundos. 3. Pulsar botón de "Opciones" y a continuación "*Stop service & exit*" 4. Comprobar el cierre de la interfaz
Notas y Preguntas:	Ninguna.

Tabla 54. Caso de Prueba 05 – Salir de la aplicación

6.2.2 Pruebas unitarias

Las pruebas unitarias son la forma de comprobar el correcto funcionamiento de cada módulo que conforma el código aisladamente. Esto asegura que cada uno de los componentes trabaja como es debido por separado. La idea será especificar un caso de prueba por cada método en el módulo de forma que sea independiente del resto. Cabe destacar sin embargo que dichas pruebas no descubrirán errores de integración, problemas que puedan surgir de rendimiento o referentes al sistema en su conjunto. La herramienta para simular el entorno de pruebas para Java será *JUnit*.

Tras configurar el proyecto mediante el entorno que provee el propio *plugin* de Eclipse para Android (véase la Figura 55), se crearían las pruebas (en un primer momento estaría vacío como indica la Figura 56) y se comprobaría su funcionamiento.

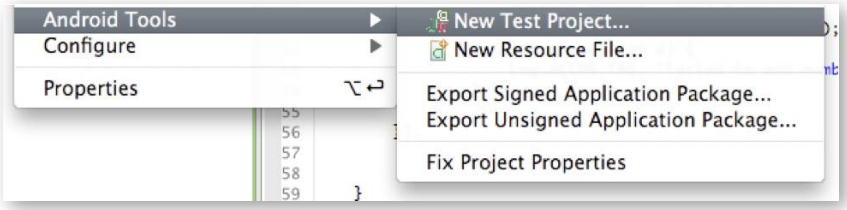


Figura 55. Pruebas unitarias – Nuevo proyecto

Figura 56. Pruebas unitarias – Proyecto vacío

Tras crear los métodos de prueba de cada una de las clases que se deseen probar [8] (véase estructura del proyecto de prueba en la Figura 57), bastaría con crear una clase extra que actúe como *Test Suite* cargando todas las pruebas para ejecución (en el caso de este TFG se ha denominado *AllTests.java*). Hecho esto, simplemente se ejecutaría o "*debuggearia*"

[8] Se ha probado únicamente la "lógica de negocio" de la aplicación, es decir, se han seleccionado solo las clases que se han creído más convenientes bajo este criterio.

dicha clase para poder apreciar que todo ha ido tal y como se esperaba. El resultado de la ejecución de las pruebas unitarias se puede apreciar en la Figura 58.

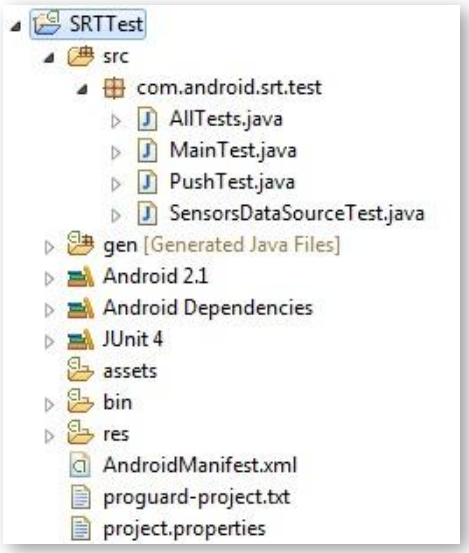


Figura 57. Pruebas unitarias – Proyecto con pruebas

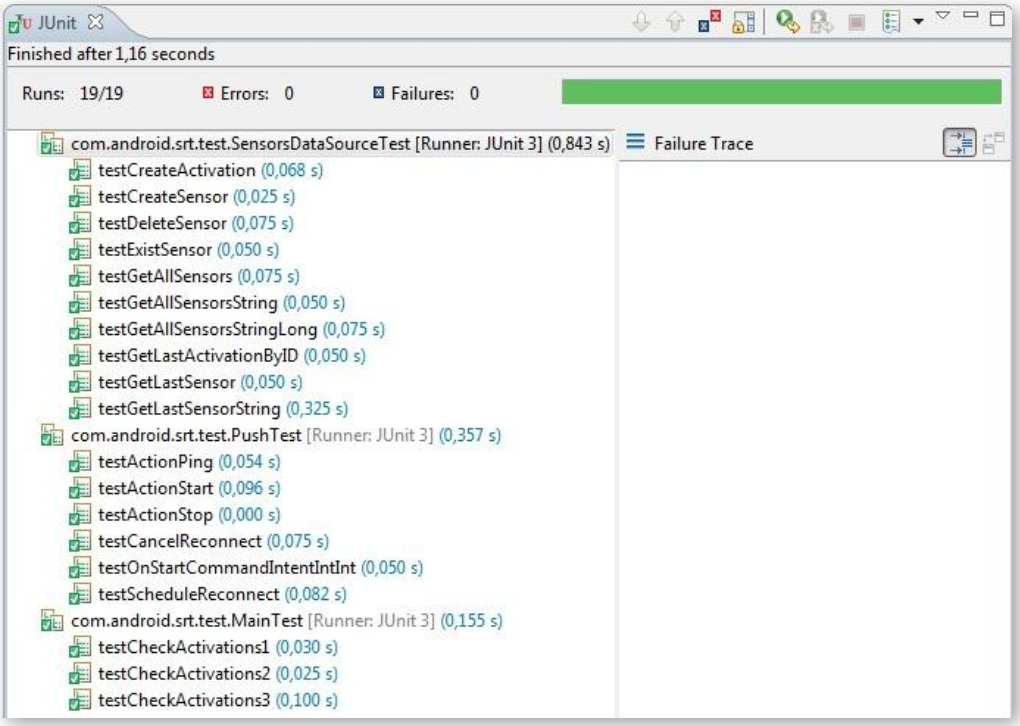


Figura 58. Pruebas unitarias – Resultado pruebas

6.2.3 Pruebas con el sistema

6.2.3.1 Evaluación del rendimiento

En este capitulo se evalúa el rendimiento de la implementación a través de una serie de experimentos para determinar si la naturaleza dinámica del sistema, resulta en penalidades de cara al rendimiento.

Para estos experimentos se empleó un hardware dedicado, tal y como se mencionó anteriormente, se han empleado *smartphones* Android para testear los clientes móviles, ya que este tipo de dispositivos puede ser fácilmente integrable con otro *software* Java. El ancho de banda bruto del enlace inalámbrico entre los clientes móviles y el sistema de publicación / suscripción se garantizó que era estable durante la evaluación.

La Figura 59 muestra el impacto del número de editores (en el ámbito de este TFG serían los sensores de la red) en el rendimiento de los mensajes (*throughput*). En este caso el experimento se llevó a cabo con un número creciente de editores y con un único suscriptor.

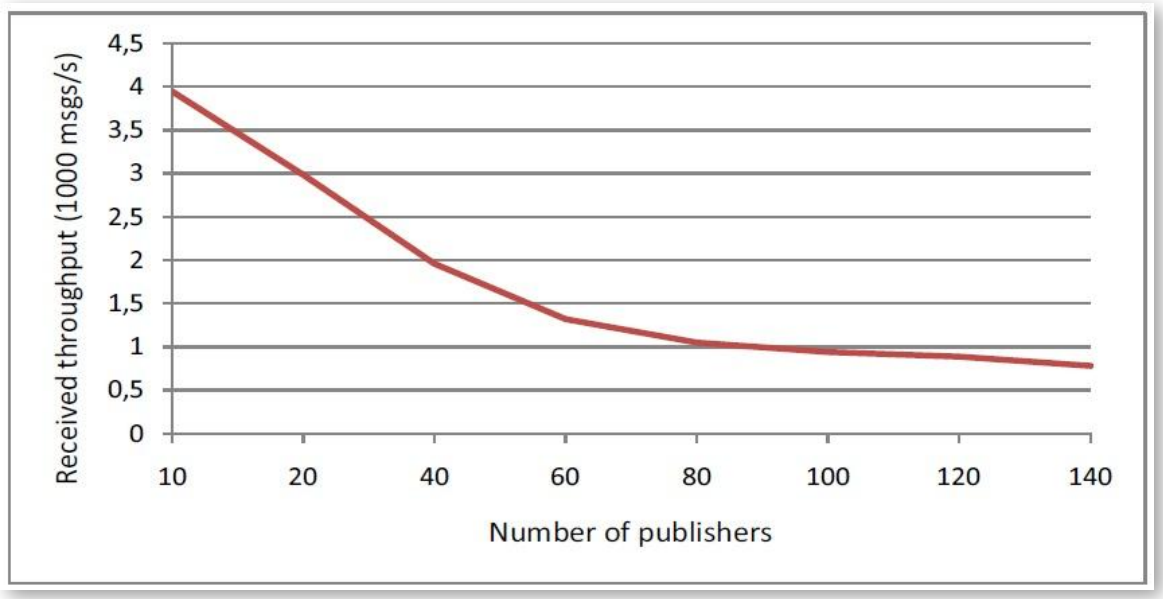


Figura 59. Impacto entre editores y rendimiento de mensajes

El sistema soporta una producción de 4000 mensajes por segundo para unos pocos editores y sobre 450 msgs/s para una cantidad de 140 editores.

De la misma también se comprobó el impacto del número de suscriptores. Ambos experimentos se repitieron al menos cinco veces. Tal y como muestra la Figura 60, la tasa de recepción de mensajes disminuye de forma significativa con un creciente número de suscriptores.

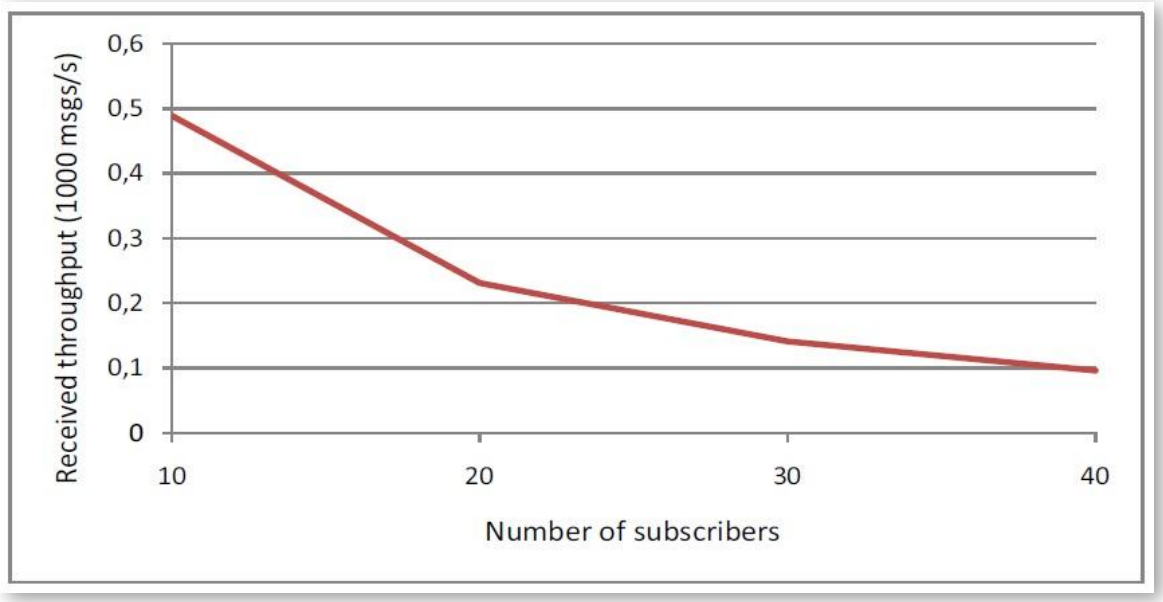


Figura 60. Impacto entre suscriptores y rendimiento de mensajes

Por último se comprobó la latencia o el tiempo requerido por un cliente en recibir un flujo de datos de un único editor, como es obvio, en una red *wireless*. La latencia es una importante métrica a considerar, especialmente donde el intercambio de datos en *soft real-time* es un requisito. Dicha gráfica sobre la latencia puede apreciarse en la Figura 61.

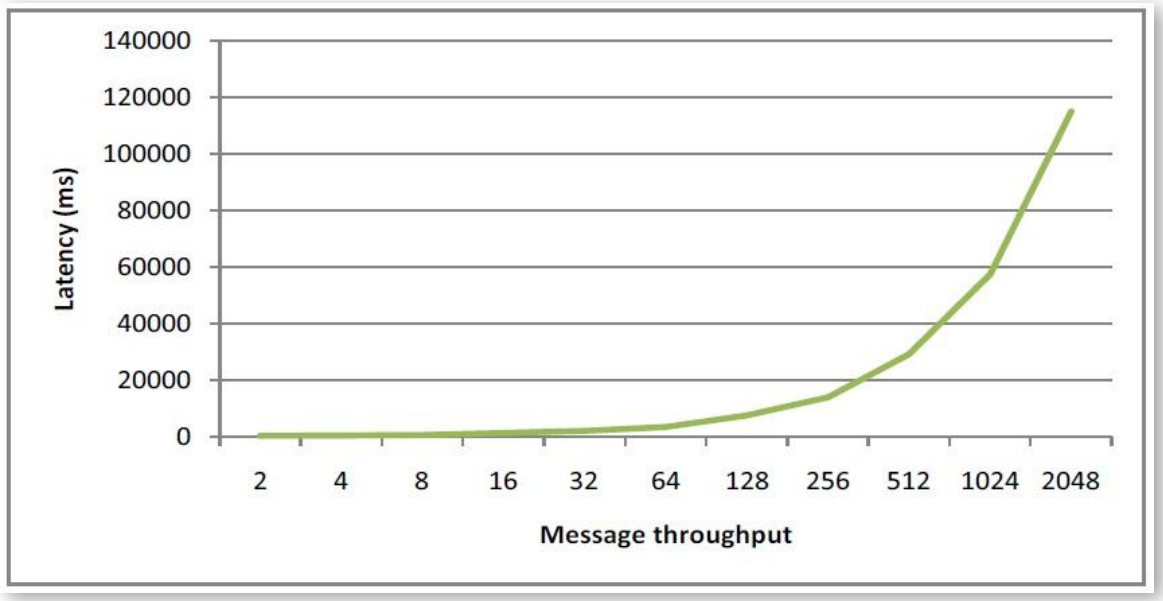


Figura 61. Tiempo de latencia para el cliente remoto MQTT. El eje x en escala logarítmica muestra el número de eventos

6.2.3.2 Compatibilidad plataformas Android

Ya se comentó con anterioridad que al estar programada la aplicación en un nivel 7 de API, se puede ejecutar en cualquier versión de la plataforma superior a la 2.1.x. Esto no limita de ninguna forma el tipo de dispositivo, bien sea *smartphone* o *Tablet*, ya que desde la versión 4 de Android se unifican ambos tipos para evitar problemas de compatibilidad.

En esta sección se comprueba la ejecución de la aplicación en un *Tablet* Asus, en comparación con como se vería en un *smartphone* tradicional (véase 6.1 "Descripción del entorno de pruebas" para ver las características concretas de estos terminales). Se sobreponen a la vez como se verían en ambos dispositivos para cada una de las acciones elementales de la aplicación, tal y como se muestra en las siguientes Figuras: Figura 62, Figura 63, Figura 64, Figura 65, Figura 66 y Figura 67.

Figura 62. Comparación app ejecución Tablet Asus / Xperia Neo V – Menú icono

Figura 63. Comparación app ejecución Tablet Asus / Xperia Neo V – Splash Screen

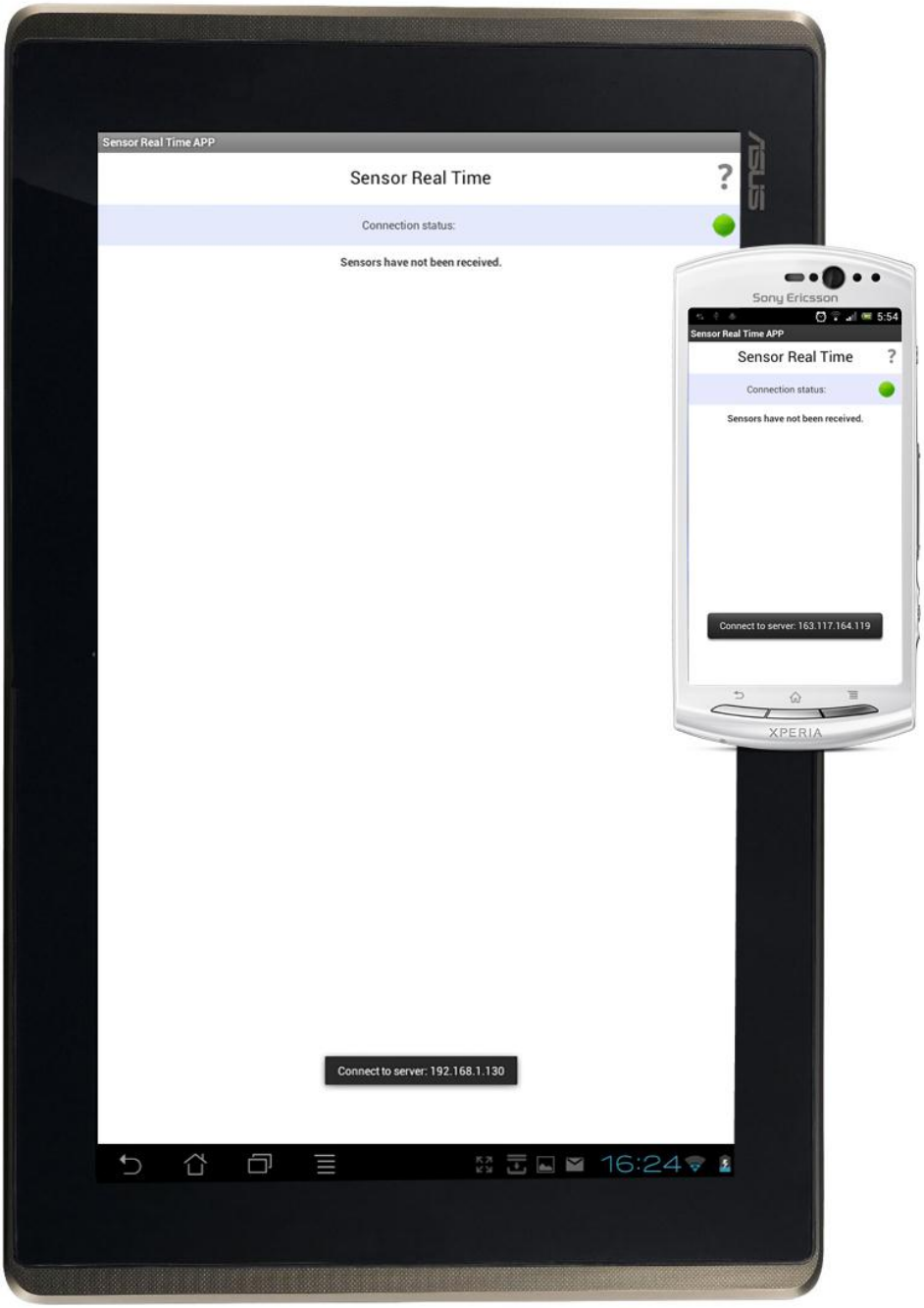


Figura 64. Comparación app ejecución Tablet Asus / Xperia Neo V – Conexión a *broker*

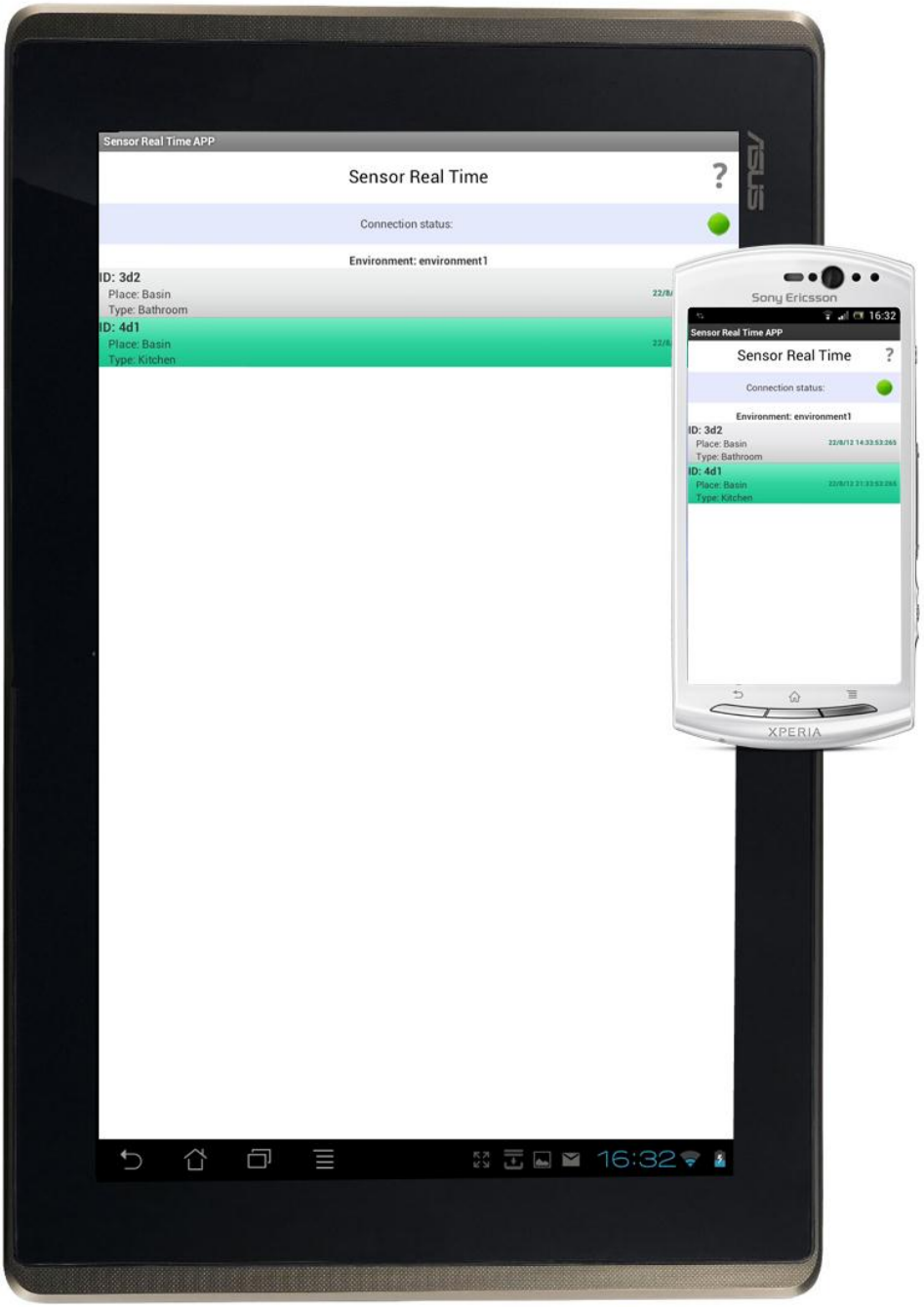


Figura 65. Comparación app ejecución Tablet Asus / Xperia Neo V – Eventos de sensores recibidos

Figura 66. Comparación app ejecución Tablet Asus / Xperia Neo V – Modificación parámetro Preferencias

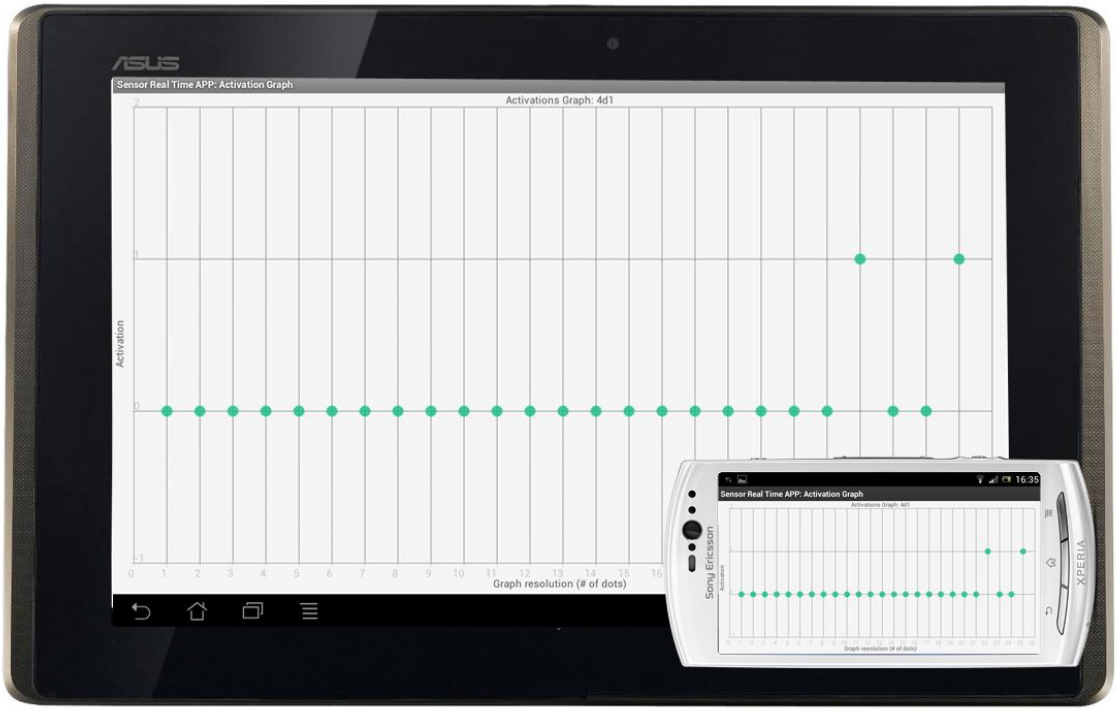


Figura 67. Comparación app ejecución Tablet Asus / Xperia Neo V – Vista de gráfica

7 Conclusiones

"Lo importante no es llegar sino ir".

Robert Louis Stevenson (1850-1894)

En este Trabajo de Fin de Grado se ha realizado el desarrollo de una aplicación para el Sistema Operativo Android enfocada en el ámbito de la telemonitorización. Dicho sistema se ha dirigido en una primera instancia para permitir la recepción de mensajes provenientes de una red de sensores. Los mensajes recibidos enviados desde la red son cada una de las activaciones producidas por un sensor de la misma. La idea era simple, recoger dichos datos para poder monitorizar el flujo de actividades de dicha red, saber que sensor concreto se había activado en que instante y lugar, y almacenarlo para un uso futuro. Este hecho de almacenar las activaciones de forma persistente, da la posibilidad de emplear los datos guardados a modo de histórico para la generación de gráficos y otras utilidades de interés.

De cara a poder codificar la aplicación para ese sistema operativo en concreto, ha sido necesario el estudio de cómo desarrollar sobre dicha plataforma, ya que a pesar de emplear Java como lenguaje base, existen clases propias y ciclos de vida a tener en cuenta. Se ha analizado a su vez, la red de sensores y el *backend* existente para desarrollar la aplicación en consecuencia. Además para conseguir el propósito que se mencionaba anteriormente, ha sido necesario profundizar en protocolos de mensajes de tipo *publish/subscribe*, más concretamente en MQTT, que permitiese al dispositivo móvil establecer una conexión con el servidor de la red para posteriormente recibir los eventos que fuesen enviados. Cada uno de los mensajes es transmitido a través de un canal al que previamente el cliente debe suscribirse, de forma lógica, solo recibirá los mensajes del canal al que esté suscrito, en nuestro caso el referente a las activaciones de sensores.

El sistema completo está desarrollado sobre el patrón arquitectónico del Modelo Vista Controlador o MVC, permitiendo separar la lógica del sistema, del acceso a los datos y de las interfaces de usuario. Esto es especialmente útil para fomentar la extensibilidad y reutilización en un sistema que de seguro seguirá aumentando en funcionalidad y características en un futuro. A esto se le añade el empleo de paquetes para restructurar la localización de las clases según su cometido dentro de la arquitectura anterior, y separar incluso, en las propias interfaces el estilo (colores, tamaño de texto, etc.) de su estructura (márgenes, ancho, alto, etc.), tal y como ocurre en *HTML* y *CSS*.

Gracias a las pruebas realizadas se ha comprobado que quedan cumplidas todas las expectativas del cliente que se vieron reflejadas en los requisitos. Se comprueba su correcto funcionamiento, y se verifican incluso requisitos que no pidió el cliente, necesarios sin embargo para el sistema, como la recepción de mensajes en *soft real-time*. Más allá de lo exigido por el cliente y del entorno concreto para el que se ha desarrollado, esta aplicación sirve de base sólida con unas ligeras modificaciones para cualquier sistema de telemonitorización básico que se plantee. Por último se ha tenido en cuenta el hecho de que los receptores iban a ser dispositivos móviles, procurando promover el mínimo consumo posible de recursos de cara a no drenar la vida de la batería (no realizar transacciones innecesarias, empleo de una base de datos ligera como SQLite, etc.).

A lo largo de la vida del proyecto se solventaron diversos problemas hasta dar con la aplicación funcional que se tiene actualmente. Se pretendió emplear el nivel 15 de API a la hora de programar, pero fue imposible hacerlo compatible con la librería de uso de MQTT, por lo que finalmente se decidió emplear el nivel 7, permitiendo de esta forma aumentar el rango de dispositivos posibles, a cualquiera con SO superior al 2.1.x de Android. También comentar que el ciclo de vida de Android supuso un problema, y más concretamente la capacidad de rotación del dispositivo a apaisado o *landscape*. Esto significó un problema, ya que el propio SO finalizaba la aplicación y la volvía a crear, pasando por alto el ciclo de vida que se había planteado para el desarrollo de la aplicación (cuándo se liberaban recursos, cuándo se cerraba la BD, etc.). Por suerte al cliente no le supuso un problema el visualizar las interfaces únicamente en *portrait* (salvo para las gráficas), por lo que se terminó incluyendo como requisito tras hablar con él.

Por último comentar que se han cumplido a raja tabla los plazos de entrega estipulados al comienzo del proyecto. Según lo que fue planificado, la desviación en tiempo para la totalidad del proyecto ha sido prácticamente nula, a pesar de que sí existen variaciones significativas entre las propias tareas, lo cual no ha supuesto ningún problema. Al final se puede considerar que el proyecto ha sido un éxito por cumplir en tiempo y recursos con lo demandado por el cliente. A título personal a mi me ha servido principalmente para aprender el funcionamiento de una tecnología actual y con una porción de cuota de mercado importante como es Android, y también por servir de simulación a un trabajo real que podría exigirse perfectamente fuera de un ámbito académico, en el que se han de cumplir ciertas pautas y exigencias en un tiempo determinado.

8 Líneas futuras

"La mejor forma de predecir el futuro es implementarlo".

Dave Heinemeier Hansson (Ruby on Rails/37signals)

El hecho de proponer futuras líneas de desarrollo implica que el sistema de telemonitorización implementado es susceptible de mejorar, aun obteniendo buenos resultados y cumpliendo todas las expectativas iniciales del cliente.

Tal y como se ha comentado numerosas veces con anterioridad, el sistema operativo enfocado en primera instancia a emplear la aplicación desarrollada, ha sido Android. Lógicamente y aun poseyendo una gran cuota de mercado, el número de dispositivos que se quedan "fuera" es demasiado alto como para no considerarlo. Es por esto que una posible mejora sería portar la aplicación a otras plataformas de renombre, como podría ser iOS y su terminal estrella iPhone.

El protocolo de comunicación empleado MQTT es *publish / subscribe*, pero lo cierto es que tanto el terminal como el servidor centralizado emplean únicamente o bien funcionalidad *publish* o *subscribe*. Es decir, el servidor se limita a realizar *publish* y el terminal móvil a suscribirse a los *topics* para recibir las posibles activaciones. Supongamos ahora un sistema algo más versátil y que cada nodo sea capaz tanto de recibir como enviar mensajes. Queda pendiente, por tanto, agregar funcionalidad para que la aplicación funcione a su vez como *publisher* o editor, que permita el envío de "telecomandos" con el fin de realizar ajusten en la configuración de los nodos de la red.

En cuanto al módulo de base de datos encargado de llevar un histórico de las activaciones de cada uno de los sensores, hay que recordar que se realizó con el fin de poder tratar a *posteriori* esos datos, mostrar gráficas, etc. Estos datos sólo se almacenaban, como es lógico, si la aplicación había estado funcionando en el momento de recibirlos. Esto a su vez implica que el espectro total de activaciones, en un día dado por ejemplo, no tiene necesariamente que coincidir con lo almacenado en la BD, si la aplicación no estuvo activa en esos momentos. Por suerte el servidor encargado de realizar los *publish* posee una BD en la que sí se registran todas las activaciones. La mejora futura sería permitir la consulta remota a dicha base de datos para poder implementar un mayor número de servicios en la aplicación.

Sería interesante traducir la aplicación a distintos idiomas aprovechando la sencillez de disponer ya de una arquitectura MVC en la que la mayoría de las cadenas de texto se encuentran en ficheros independientes.

Por último y no menos importante, habría que tener en cuenta que a la hora de desplegar el sistema en un entorno real con los datos personales de las personas residentes en el domicilio, el flujo de mensajes debería ser seguro. La última mejora de esta lista sería, por tanto, añadir una capa de seguridad al protocolo de comunicación entre aplicación y servidor, ya que por el momento los datos van en "claro".

Glosario de términos

ESA	*Agencia Espacial Europea*
RU	*Requisito de Usuario*
RS	*Requisito de Software*
SW	*Software*
SO	*Sistema Operativo*
MQTT	*Message Queue Telemetry Transport*
TFG	*Trabajo de fin de Grado*
(G)UI	*(Graphic) User Interface*
API	*Application Programming Interface*
SDK	*Software Development Kit*
QoS	*Quality of Service*
IDE	*Integrated Development Environment*
CU	*Caso de Uso*
UML	*Unified Model Language*
MVC	*Modelo Vista Controlador*

Bibliografía

[1] Intel y los nuevos transistores Tri-Gate '3D' de sus Ivy Bridge
http://www.xataka.com/componentes-de-pc/intel-y-los-nuevos-transistores-tri-gate-3d-de-sus-ivy-bridge
[08 de Mayo de 2011]
Último acceso: Junio 2012

[2] Ray Kurzweil sobre cómo la tecnología nos transformará
http://www.ted.com/talks/lang/es/ray_kurzweil_on_how_technology_will_transform_us.html.
[Noviembre 2006]
Último acceso: Junio 2012

[3] Mobile apps developers in hot demand
http://www.theaustralian.com.au/business/breaking-news/mobile-apps-developers-in-hot-demand/story-e6frg90f-1226338906327
[26 de Abril de 2012]
Último acceso: Junio 2012

[4] Modelos de desarrollo de software: Cascada vs V
http://ddonofrio.blogspot.com.es/2010/12/modelos-de-desarrollo-de-software.html
[25 de Diciembre de 2010]
Último acceso: Junio 2012

[5] Ciclo de vida del Software y Modelos de desarrollo
http://www.cepeu.edu.py/LIBROS_ELECTRONICOS_3/lpcu097%20-%2001.pdf
Instituto de Formación Profesional - Libros Digitales. pp. 21-24

[6] Leyes de evolución del Software
http://cnx.org/content/m17406/latest/
Connexions – Educational content repository
[24 de Noviembre 2008]
Último acceso: Junio 2012

[7] Guía para la aplicación de Estándares de Ingeniería de Software ESA
http://www.ie.inf.uc3m.es/grupo/docencia/reglada/Is1y2/ESA/BSSC962-ES.PDF
ESA Comité de Estandarización y Control de Software (BSSC). pp. 4-5

[8] Matriz de Trazabilidad
http://www.softqatest.net/?p=77
[15 de Junio de 2010]
Último acceso: Junio 2012

[9] Acoplamiento y cohesión
http://latecladeescape.com/articulos/1547-acoplamiento-y-cohesion
[11 de Noviembre de 2006]
Último acceso: Julio 2012

[10] El modelo MVC de JAVA
http://guillermoarreola.blogspot.es/
[27 de Julio de 2009]
Último acceso: Julio 2012

[11] Bev Littlewood and Lorenzo Strigini
The Risks of Software. *Scientific* American. Volumen 268, Número 1, Enero, 1993

[12] Y. Huang, P. Jalote and C. Kintala
Two Techniques for Transient Software Error Recovery. Lecture Notes in *Computer Science*, Vol. 774, pp. 159-170, 1994

[13] La máquina virtual Dalvik
http://androideity.com/2011/07/07/la-maquina-virtual-dalvik/
[07 de Julio de 2011]
Último acceso: Julio 2012

[14] App Runner
http://www.netmite.com/android/
[07 de Febrero de 2012]
Último acceso: Julio 2012

[15] Android Supported Media Formats
http://developer.android.com/guide/appendix/media-formats.html
[Julio 2012]
Último acceso: Julio 2012

[16] Use Your Android Phone as a Wireless Modem
http://developer.android.com/resources/articles/speech-input.html
[06 de Mayo de 2010]
Último acceso: Julio 2012

[17] Announcing the Android 1.0 SDK, release 1
http://android-developers.blogspot.com.es/2008/09/announcing-android-10-sdk-release-1.html
[23 de Septiembre de 2008]
Último acceso: Julio 2012

[18] Android 1.1 SDK, release 1 Now Available
http://android-developers.blogspot.com.es/2009/02/android-11-sdk-release-1-now-available.html
[09 de Febrero de 2009]
Último acceso: Julio 2012

[19] Why does Google name its Android products after desserts?
http://articles.cnn.com/2011-02-04/tech/google.honeycomb.android.names_1_google-android-android-os-randall-sarafa?_s=PM:TECH
[04 de Febrero]
Último acceso: Julio 2012

[20] Google Now
http://www.google.com/landing/now/
[Julio 2012]
Último acceso: Julio 2012

[21] Current Distribution | Platform Versions
http://developer.android.com/about/dashboards/index.html
[2 de Julio de 2012]
Último acceso: Julio 2012

[22] On Android Compatibility
http://android-developers.blogspot.com.es/2010/05/on-android-compatibility.html
[31 de Mayo de 2010]
Último acceso: Julio 2012

[23] Google´s Strong Mobile-Related Patent Portfolio
http://paidcontent.org/tech/419-googles-strong-mobile-related-patent-portfolio/
[21 de Septiembre de 2007]
Último acceso: Julio 2012

[24] Is the Google Phone an Unauthorized Replicant?
http://bits.blogs.nytimes.com/2009/12/15/is-the-google-phone-an-unauthorized-replicant/
[15 de Diciembre de 2009]
Último acceso: Julio 2012

[25] Bada se transformará en Tizen
http://www.genbeta.com/sistemas-operativos/bada-se-transformara-en-tizen
[15 de Enero de 2012]
Último acceso: Julio 2012

[26] Microsoft and Nokia Partner on Smartphone Future
http://www.pcworld.com/article/219416/microsoft_and_nokia_partner_on_smartphone_future.html
[11 de Febrero de 2011]
Último acceso: Julio 2012

[27] iOS 5 vs. Android 4.0 Ice Cream Sandwich vs. Windows Phone 7.5 Mango – Comparison
http://www.redmondpie.com/ios-5-vs.-android-4.0-ice-cream-sandwich-vs.-windows-phone-7.5-mango-comparison/
[22 de Octubre de 2011]
Último acceso: Julio 2012

[28] Mobile Operating Systems Compared: iOS, Android and Windows Phone
http://community.giffgaff.com/t5/Blog/Mobile-Operating-Systems-Compared-iOS-Android-and-Windows-Phone/ba-p/2776337
[18 de Enero de 2012]
Último acceso: Julio 2012

[29] Mobile OS Showdown: Android, BlackBerry, iOS, and Windows Phone 7
http://www.pcworld.com/businesscenter/article/229173/mobile_os_showdown_android_blackberry_ios_and_windows_phone_7.html
[02 de Junio de 2011]
Último acceso: Julio 2012

[30] 64 million Smart phones shipped worldwide in 2006
http://www.canalys.com/newsroom/64-million-smart-phones-shipped-worldwide-2006
[12 de Febrero de 2007]
Último acceso: Julio 2012

[31] Gartner Says Sales of Mobile Devices Grew 5.6 Percent in Third Quarter of 2011; Smartphone Sales Increased 42 Percent
http://www.gartner.com/it/page.jsp?id=1848514
[15 de Noviembre de 2011]
Último acceso: Julio 2012

[32] Two Thirds of New Mobile Buyers Now Opting For Smartphones
http://blog.nielsen.com/nielsenwire/online_mobile/two-thirds-of-new-mobile-buyers-now-opting-for-smartphones/
[12 de Julio de 2012]
Último acceso: Agosto 2012

[33] Más del 72% de los dispositivos vendidos en España llevan Android
http://www.xatakandroid.com/mercado/mas-del-72-de-los-dispositivos-vendidos-en-espana-llevan-android
[15 de Mayo de 2012]
Último acceso: Julio 2012

[34] Gráfico IDC Q2 2012 elaborado por 9to5Mac
http://9to5mac.files.wordpress.com/2012/07/n.jpg
Último acceso: Julio 2012

[35] Android Expected to Reach Its Peak This Year as Mobile Phone Shipments Slow, According to IDC
http://www.sys-con.com/node/2291309
[06 de Junio de 2012]
Último acceso: Julio 2012

[36] Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.:
The many faces of publish/subscribe (2003)

[37] Cugola, G., Murphy, A.L., Picco
Content-based Publish-subscribe in a Mobile Environment.
Bellavista, P., Corradi, A. (eds.) Mobile Middleware, pp. 257–285. Auerbach Publications (2006), invited contribution

[38] Shnayder, V., Chen, B.r., Lorincz, K., Jones, T.R.F.F., Welsh, M.: Sensor networks for medical care. In: Proceedings of the 3rd international conference on Embedded networked sensor systems. pp. 314–314.

[39] Fiege, L., Grtner, F.C., Kasten, O., Zeidler, A.:
Supporting mobility in contentbased publish/subscribe middleware (2003)

[40] Fiege, L., Mhl, G., Grtner, F.C.:
Modular event-based systems.
THE KNOWLEDGE ENGINEERING REVIEW 17, 359–388 (2006)

[41] Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.:
The many faces of publish/subscribe (2003)

[42] Huang, Y., Garcia-Molina, H.:
Publish/subscribe in a mobile environment.
Wirel. Netw. 10, 643–652 (November 2004),
http://dx.doi.org/10.1023/B:WINE.0000044025.64654.65

[43] Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., Anderson, J.:
Wireless sensor networks for habitat monitoring.
In: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications. pp. 88–97. WSNA '02, ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/570738.570751

[44] MQ Telemetry Transport
http://mqtt.org/
Último acceso: Agosto 2012

[45] Chen, D., Varshney, P.K.: Qos support in wireless sensor networks: A survey. In: International Conference on Wireless Networks'04. pp. 227–233 (2004)

[46] Hunkeler, U., Truong, H.L., Stanford-Clark, A.: MQTT-S – a publish/subscribe protocol for wireless sensor networks. In: Proceedings of the 2nd Workshop on Information Assurance for Middleware Communications (IAMCOM'08) (Jan 2008)

[47] van Kasteren, T.L.M., Kröse, B.J.: A sensing and annotation system for recording datasets in multiple homes. In: Proceedings of the CHI 2009 Workshop on Developing Shared Home Behavior Datasets to Advance HCI and Ubiquitous Computing Research. Boston, USA (2009)

[48] Google Play | Media Remote for Android
https://play.google.com/store/apps/details?id=com.sony.seconddisplay.view&hl=es
Último acceso: Agosto 2012

[49] Apps: MyRemote App for Android
http://www.supportforum.philips.com/en/forumdisplay.php?72-MyRemote-for-Android
Último acceso: Agosto 2012

[50] AZ Remote
http://www.octaval.com/web/products/azremote/
Último acceso: Agosto 2012

[51] EagleEyes
http://vigimax.com/pdf/manual-del-software-android.pdf
Último acceso: Agosto 2012

[52] D–Link | Notas de Prensa
http://www.dlink.es/cs/Satellite?c=Press_C&childpagename=DLinkEurope-ES%2FDLPressRelease&cid=1197392326646&p=1197318960445&packedargs=locale%3D1195806681347&pagename=DLinkEurope-ES%2FDLWrapper
[15 de Diciembre de 2011]
Último acceso: Agosto 2012

[53] Exploring the SDK
http://developer.android.com/sdk/exploring.html
[Julio 2012]
Último acceso: Julio 2012

[54] OEM USB Drivers
http://developer.android.com/tools/extras/oem-usb.html#Drivers
[Julio 2012]
Último acceso: Julio 2012

[55] Hard and Soft Real–Time
http://docs.fedoraproject.org/en-US/Fedora/15/html/Musicians_Guide/sect-Musicians_Guide-Hard_and_Soft_Real_Time.html
Último acceso: Agosto 2012

[56] Don´t Repeat Yourself
http://c2.com/cgi/wiki?DontRepeatYourself
[13 de Diciembre de 2011]
Último acceso: Agosto 2012

[57] Data Access Object Design Pattern
http://www.codefutures.com/data-access-object/
Último acceso: Septiembre 2012

[58] Supporting Multiple Screens
http://developer.android.com/guide/practices/screens_support.html
Último acceso: Septiembre 2012

[59] android.support.v4.app
http://developer.android.com/reference/android/support/v4/app/package-summary.html
Último acceso: Septiembre 2012

[60] Holo Everywhere
http://android-developers.blogspot.com.es/2012/01/holo-everywhere.html
[03 de Enero de 2012]
Último acceso: Septiembre 2012

Anexos

Anexo A: Manual de usuario

A continuación se expone de forma muy breve y ayudándose en base a capturas de la aplicación, un posible manual de usuario. Como requisito mínimo debería disponerse de un dispositivo Android con versión superior o igual a la 2.1.x. Tras instalar el .apk de la aplicación, lo primero sería obviamente acceder a la misma (véase Figura 69), lo cual cargaría por defecto una pantalla de bienvenida con 5 segundos de duración (al tocar la pantalla se quitaría) tal y como muestra la Figura 68, que daría paso a la interfaz principal del sistema.

Figura 69. Manual de Usuario – Icono

Figura 68. Manual de Usuario – *Splash Screen*

Acto seguido el sistema trataría de conectarse con el servidor (véase Figura 71), según los parámetros establecidos por defecto de puerto, *ip*, y suscripción de *topic* (todos ellos configurables como se ve posteriormente). En caso de no poder realizarse la conexión el indicador aparecería en rojo. Una vez conectado y suscrito al *topic* correcto para recibir eventos, el dispositivo ya está listo para la recepción de activaciones. A medida que se reciben activaciones, se actualiza la lista de sensores (filtrada por "entorno") resaltando con un fondo distinto (verde-azulado) el último sensor en activarse (véase Figura 70).

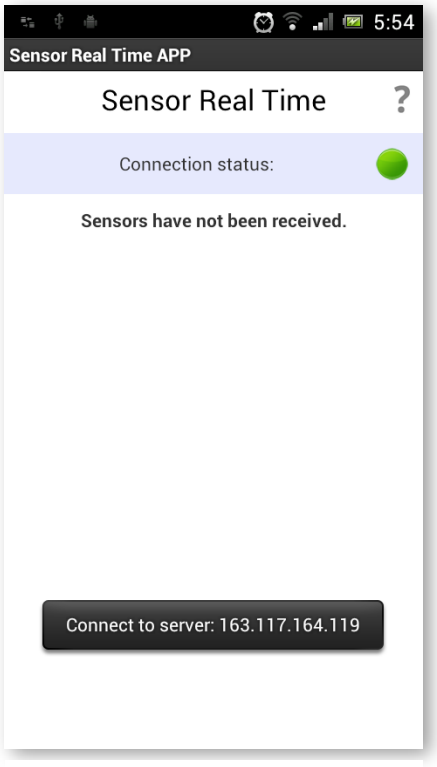


Figura 71. Manual de Usuario – Conexión

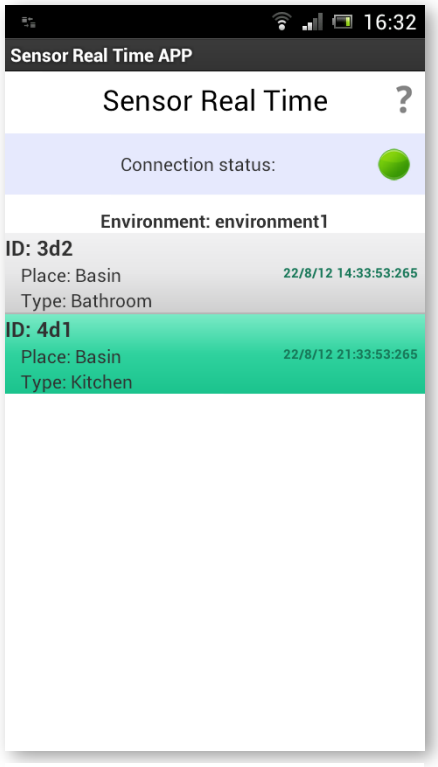


Figura 70. Manual de Usuario – Recepción de activaciones

Al pulsar la tecla de opciones (bien sea desde la pantalla o desde la tecla "hardware" en caso de tenerla) aparecería un menú con 2 opciones (véase Figura 72). La primera opción detiene el servicio y finaliza la aplicación, la segunda opción muestra la lista de preferencias con los parámetros del sistema que se pueden modificar (véase Figura 73).

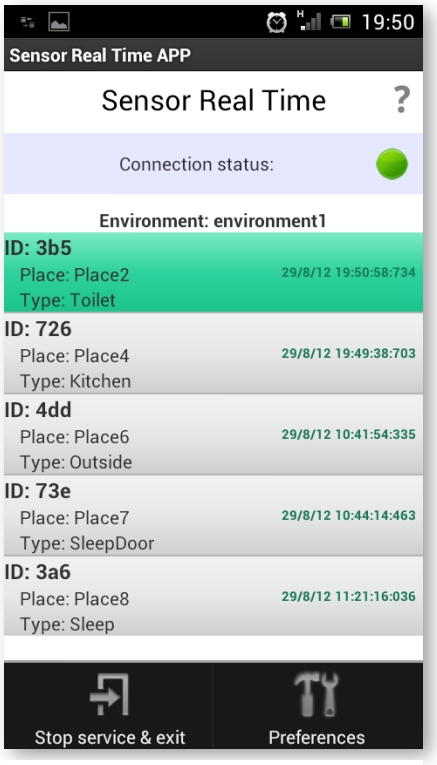


Figura 72. Manual de Usuario – Menú

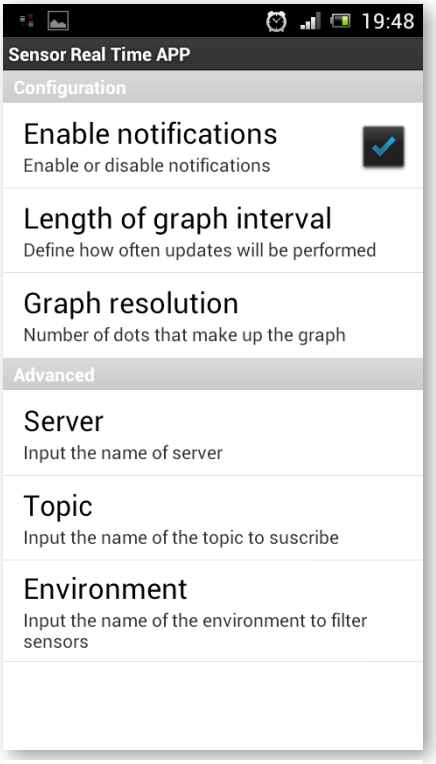


Figura 73. Manual de Usuario – Preferencias

Desde la interfaz inicial también es posible mostrar la información actual sobre la conexión al pulsar sobre el icono **?** (véase Figura 74) y mostrar una gráfica de activaciones al pinchar sobre uno de los sensores de la lista (véase Figura 75). Para facilitar la visualización en resoluciones de puntos (parámetro configurable) altas, se permite el modo *landscape* al rotar el terminal, tal y como muestra la Figura 76. Hasta aquí lo referente al manual.

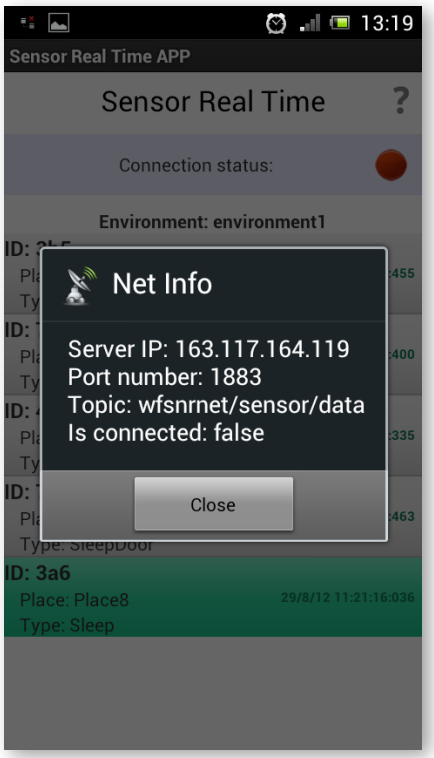


Figura 74. Manual de Usuario – Información conexión

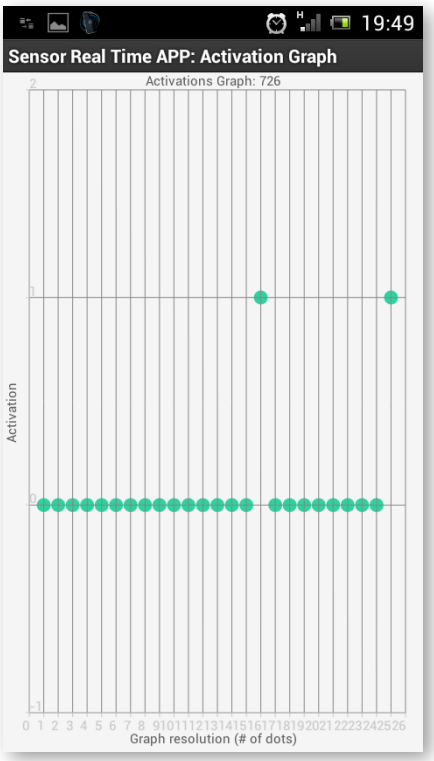


Figura 75. Manual de Usuario – Gráfica sensor

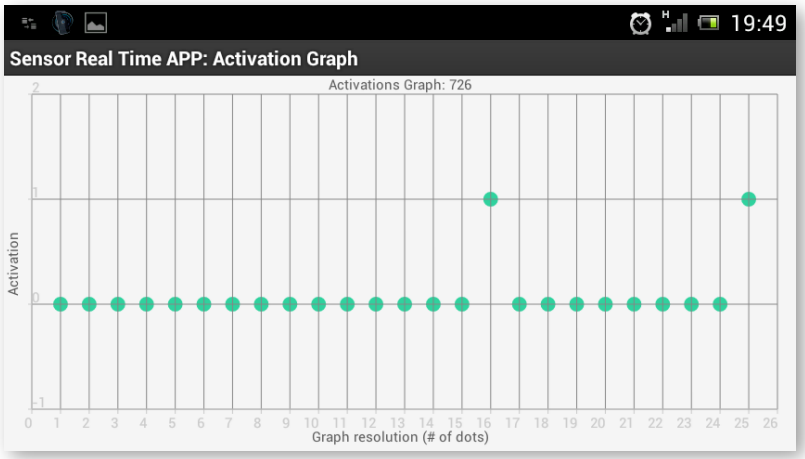


Figura 76. Manual de Usuario – Gráfica sensor *landscape*

Anexo B: Planificación

A la hora de realizar la imputación de horas se ha decidido realizar de forma realista, es decir, en lugar de imputar a cada una de las tareas su recurso correspondiente en un equipo simulado (horas hombre de diseñador a la tarea de diseño, etc.), se ha optado por que el alumno a cargo del proyecto (Juan Rubio) haga la función de cada uno de los distintos recursos, tal y como está ocurriendo realmente. No obstante a la hora de presentar el presupuesto (véase Anexo C), se tendrá en cuenta el costo en función de las horas utópicas desempeñadas por cada uno de los distintos roles.

Se ha definido a su vez una jornada laboral como una de 6 horas de trabajo en lugar de las 8 comunes, lo que hace un total de 30 horas a la semana. Tras generar las tareas del proyecto y asignárselas al único recuso existente, se obtiene el desglose de trabajo mostrado en la Figura 77.

	Nombre del recurso	Trabajo
1	Juan Rubio Iradi	378 horas
	Toma de contacto Android	*6 horas*
	Análisis del problema	*9 horas*
	Redacción de requisitos	*6 horas*
	Casos de uso	*9 horas*
	Arquitectura de la aplicación	*6 horas*
	Desarrollo de diagramas	*18 horas*
	Desarrollo de la Interfaz	*24 horas*
	Desarrollo de la lógica de nego	*120 horas*
	Testing	*9 horas*
	Introducción	*6 horas*
	Gestión y organización	*12 horas*
	Estado de la cuestión	*30 horas*
	Objetivos	*6 horas*
	Diseño e Implementación	*48 horas*
	Pruebas y Evaluación	*9 horas*
	Resumen y Abstract	*12 horas*
	Conclusiones	*12 horas*
	Líneas Futuras	*12 horas*
	Anexos	*18 horas*
	Agradecimientos	*6 horas*

Figura 77. Trabajo (en horas) asignado al recurso

Además para las tareas que se realizan en paralelo se ha decidido disminuir la carga de cada una individualmente de la siguiente forma:

Horas de la jornada / nº de tareas solapadas

Así para 2 tareas solapadas cada una tendría un trabajo de 3 horas.

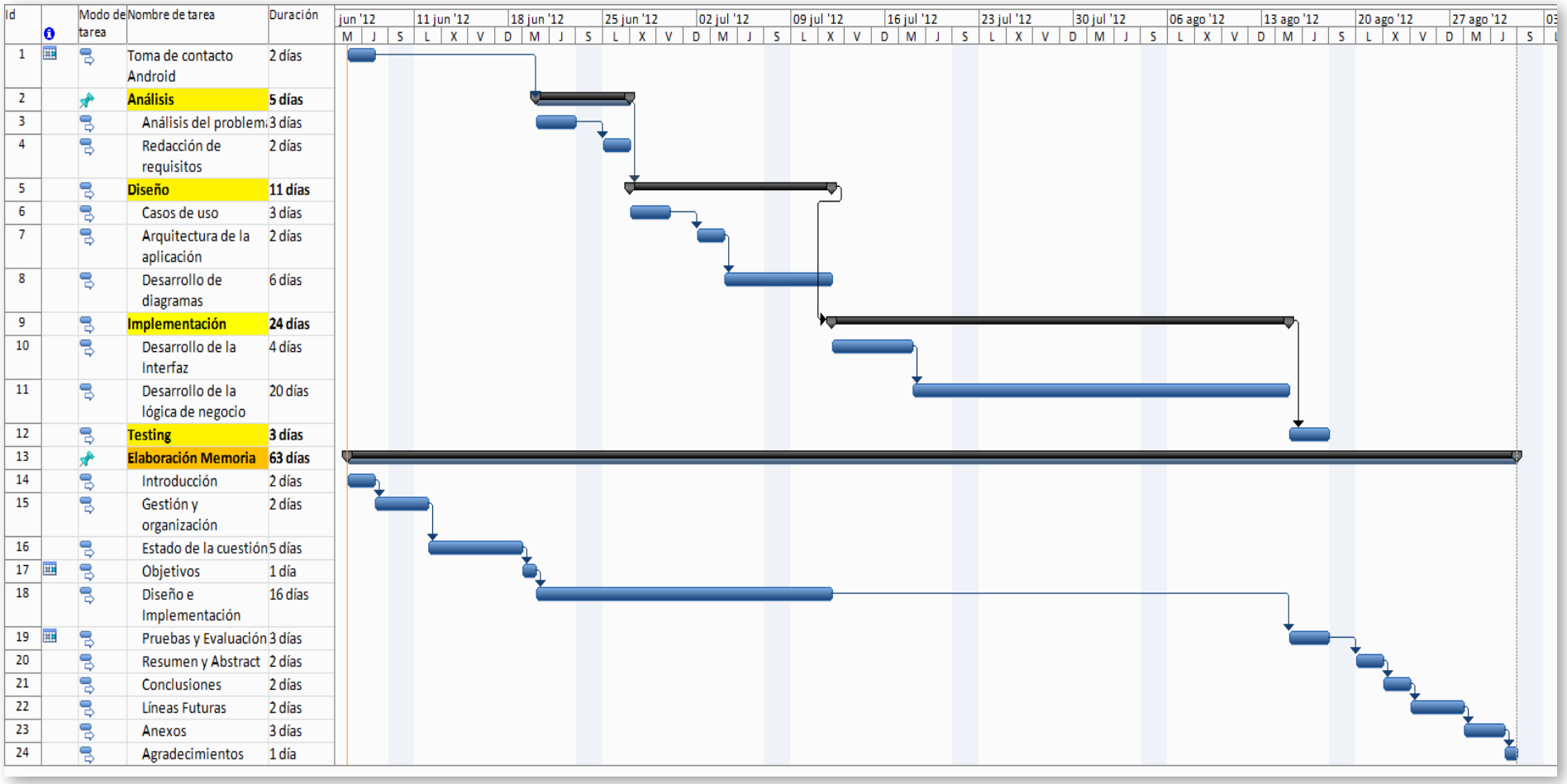
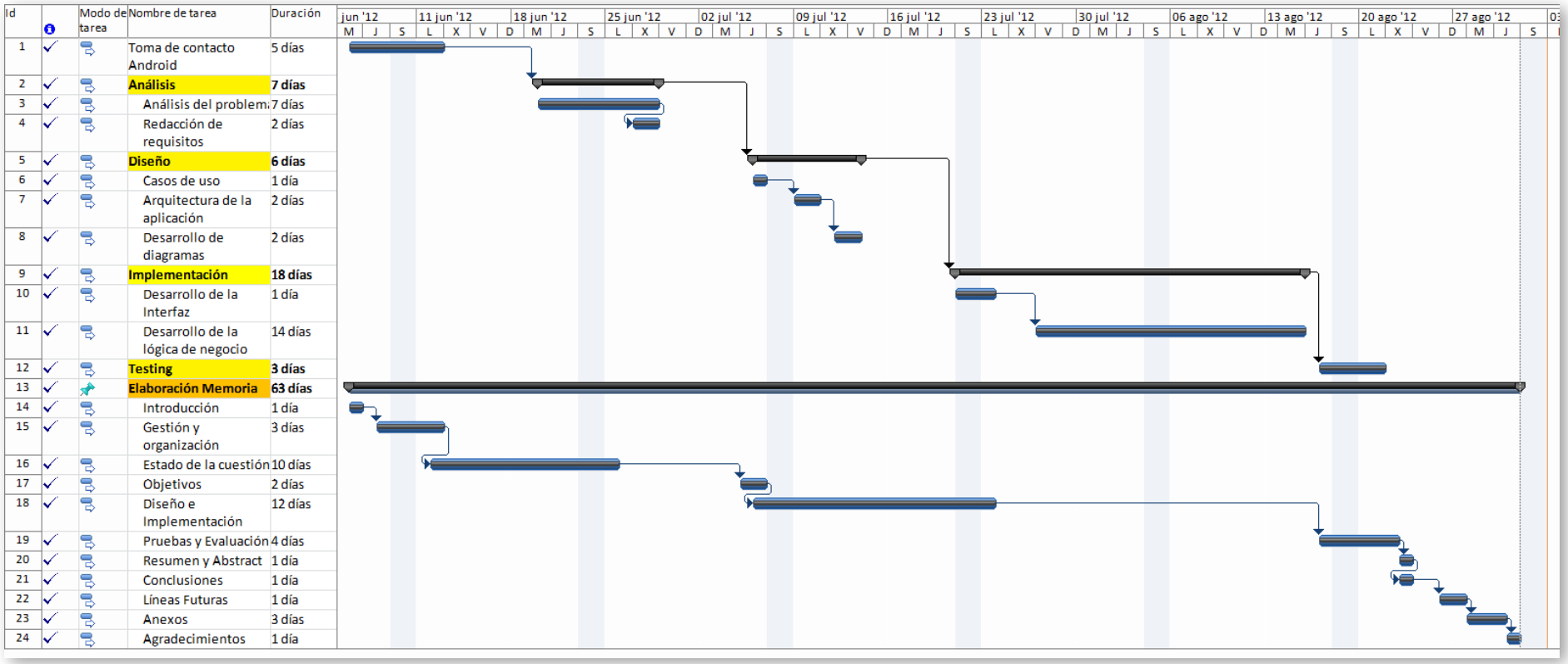


Figura 78. Planificación inicial



Id	Modo de tarea	Nombre de tarea	Duración
1		Toma de contacto Android	5 días
2		**Análisis**	**7 días**
3		Análisis del problem	7 días
4		Redacción de requisitos	2 días
5		**Diseño**	**6 días**
6		Casos de uso	1 día
7		Arquitectura de la aplicación	2 días
8		Desarrollo de diagramas	2 días
9		**Implementación**	**18 días**
10		Desarrollo de la Interfaz	1 día
11		Desarrollo de la lógica de negocio	14 días
12		**Testing**	**3 días**
13		**Elaboración Memoria**	**63 días**
14		Introducción	1 día
15		Gestión y organización	3 días
16		Estado de la cuestión	10 días
17		Objetivos	2 días
18		Diseño e Implementación	12 días
19		Pruebas y Evaluación	4 días
20		Resumen y Abstract	1 día
21		Conclusiones	1 día
22		Líneas Futuras	1 día
23		Anexos	3 días
24		Agradecimientos	1 día

Figura 79. Tiempo real dedicado

En la Figura 78 se muestra la planificación inicial realizada al comienzo del proyecto en contraposición con el tiempo real dedicado tal y como muestra la imagen de la Figura 79. Esta desviación en tiempo se aprecia mejor en la tabla reflejada de la Figura 80.

	Nombre del recurso	Trabajo	Diferencia (Plan - Real)
1	Juan Rubio Iradi	376 horas	+2
	Toma de contacto Android	*30 horas*	*-24*
	Análisis del problema	*41 horas*	*-32*
	Redacción de requisitos	*6 horas*	*0*
	Casos de uso	*2 horas*	*+7*
	Arquitectura de la aplicación	*6 horas*	*0*
	Desarrollo de diagramas	*6 horas*	*+12*
	Desarrollo de la Interfaz	*3 horas*	*+21*
	Desarrollo de la lógica de nego	*84 horas*	*+36*
	Testing	*9 horas*	*0*
	Introducción	*3 horas*	*+3*
	Gestión y organización	*20 horas*	*+8*
	Estado de la cuestión	*70 horas*	*-40*
	Objetivos	*5 horas*	*+1*
	Diseño e Implementación	*40 horas*	*+8*
	Pruebas y Evaluación	*15 horas*	*-6*
	Resumen y Abstract	*3 horas*	*+9*
	Conclusiones	*3 horas*	*+9*
	Líneas Futuras	*6 horas*	*+6*
	Anexos	*18 horas*	*0*
	Agradecimientos	*6 horas*	*0*

Figura 80. Trabajo (en horas) realizado por el recurso

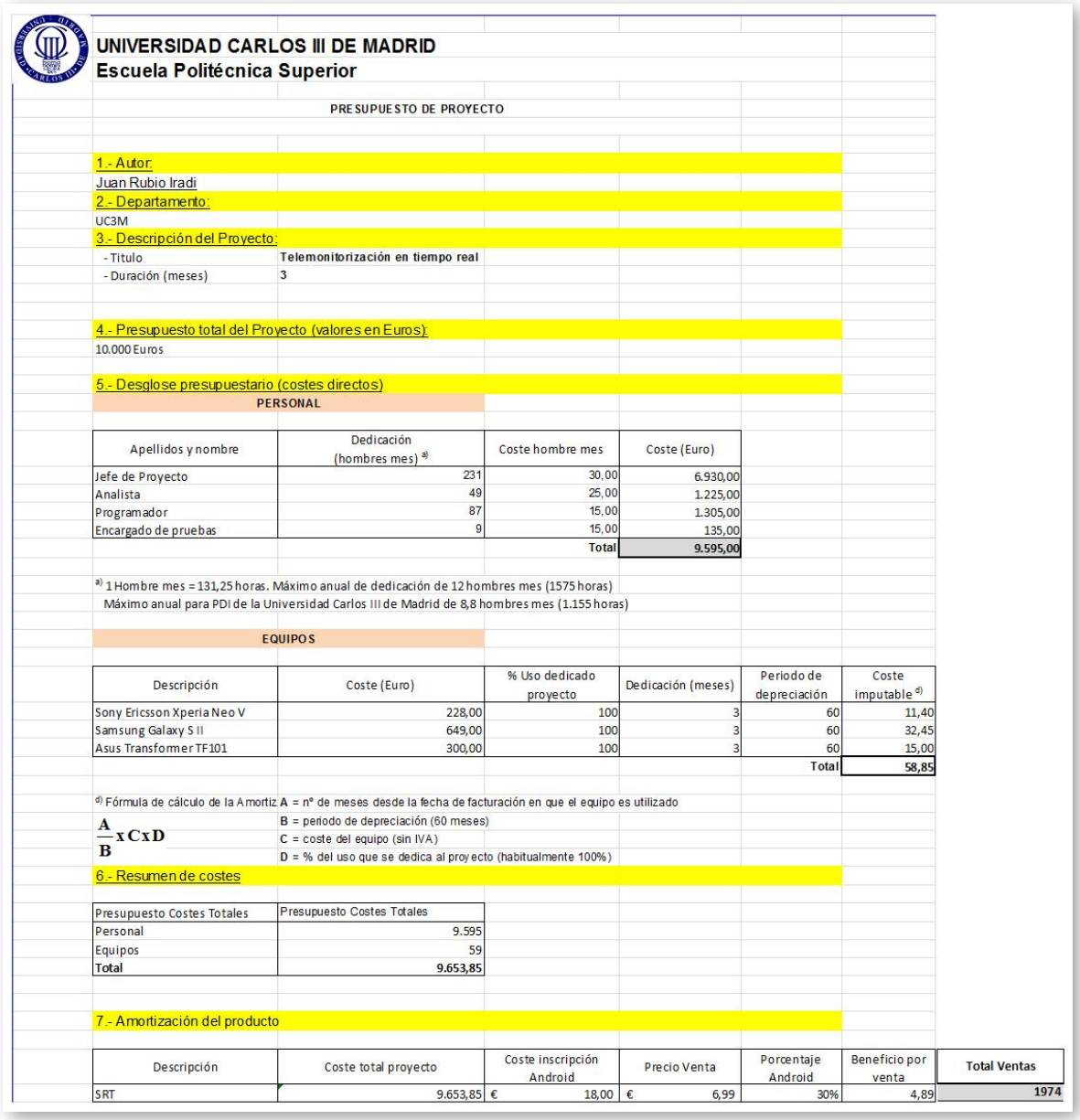
Tal y como puede apreciarse la desviación respecto a la planificación inicial es insignificante, apenas un par de horas, pero si se observa detenidamente si se aprecian desviaciones importantes entre cada una de las tareas a realizar.

La diferencia en positivo indica que la planificación se realizó de manera algo holgada, es decir, la tarea se cumplió en un plazo menor del esperado, sin embargo los valores negativos indican que el trabajo real superó en momentos al planificado. Por ejemplo se invirtió más tiempo en la "Toma de contacto con Android" del previsto, pero a la hora de desarrollar la lógica de negocio se dedicaron 36 horas menos de las estimadas.

Anexo C: Presupuesto

Finalmente se muestra en este anexo el presupuesto que ha implicado el desarrollo del sistema de telemonitorización, describiendo los gastos del proyecto, el personal de desarrollo y los equipos empleados para ello. Dicho presupuesto puede encontrarse en la Figura 81.

El presupuesto total de este proyecto asciende a la cantidad de **9.653,85 EUROS**.



UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
Juan Rubio Iradi
2.- Departamento:
UC3M
3.- Descripción del Proyecto:

- Titulo	**Telemonitorización en tiempo real**
- Duración (meses)	3

4.- Presupuesto total del Proyecto (valores en Euros):
10.000 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)
Jefe de Proyecto	231	30,00	6.930,00
Analista	49	25,00	1.225,00
Programador	87	15,00	1.305,00
Encargado de pruebas	9	15,00	135,00
		Total	**9.595,00**

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Sony Ericsson Xperia Neo V	228,00	100	3	60	11,40
Samsung Galaxy S II	649,00	100	3	60	32,45
Asus Transformer TF101	300,00	100	3	60	15,00
				Total	**58,85**

[d] Fórmula de cálculo de la Amortiz: $\frac{A}{B} \times C \times D$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

6.- Resumen de costes

Presupuesto Costes Totales	Presupuesto Costes Totales
Personal	9.595
Equipos	59
Total	**9.653,85**

7.- Amortización del producto

Descripción	Coste total proyecto	Coste inscripción Android	Precio Venta	Porcentaje Android	Beneficio por venta	Total Ventas
SRT	9.653,85	€ 18,00	€ 6,99	30%	4,89	1974

Figura 81. Presupuesto del proyecto